PC-9801 スーパーテクニック

小高輝真・清水和文・速水祐 共著

アスキー出版局

PC-9801
スーパーテクニック

小高輝真・清水和文・速水祐　共著

アスキー出版局

著者序文

PC-9800シリーズは良くも悪くも日本のパソコンの代名詞である。そのアーキテクチャに整然さはなく、10年というパソコンにしては長い年月によってさらに複雑なものになっている。しかし、さまざまな問題点を抱えているとはいえ、事務処理、ワードプロセッサ、通信、その他多くの仕事に使用されつづけているパソコンであることに変わりはない。

PC-9800シリーズ上の実用的なプログラムの多くは、使い易い操作環境を提供するため、ハードウェアに密着した工夫をほどこしている。ソフトウェアがハードウェアに密着すればするほど、ソフトウェアの移植性は低下する。しかし、ハードウェアを使いこなせば、よりきめの細かいユーザーインターフェースが実装でき、高速な処理が可能になるのである。

しかし、ハードウェアを使いこなそうとすると、PC-9800シリーズのハードウェアと8086系CPUのソフトウェアについて、広範な知識が必要となる。『PC-9800シリーズテクニカルデータブック』(アスキー出版局刊)や各デバイスの仕様書だけで、具体的なプログラムに結びつけるのは容易なことではない。

結局は何回かの試行錯誤の上に、修得するしかない。我々も例外ではない。やはり、こうした道をたどってきたのだ。本書は、まさにその試行錯誤の集大成といえる。本書では、PC-9801のハードウェアを活用する上で資料だけでは理解しづらい点や具体的なハードウェアへのアクセス方法を、プログラムを交えながら解説した。

たとえば、グラフィックのプログラミングは、速度が求められるため、ハードウェアに密着したプログラミング技術が要求される。本書では、あえてグラフLIOを使用せずに、PC-9800シリーズに備わるGDC、GRCG、EGCを直接操作して高速なプログラムを作る方法を解説した。

また、実用的な通信ソフトを作ろうとすると、必ずシリアルポートを独自に制御するハードウェア割り込み処理を書かざるをえない。このようなハードウェア割り込み処理を作成する場合の手助けとなるように、体系的な解説よりもプログラムリストを示した実践的な手法を解説した。

このように、本書は実践的なテクニックの宝庫といえる。さらに、資料編にシステム共通域やMS-DOSのワークエリア、割り込み、I/Oポートの一覧を用意した。これらは、PC-9800シリーズの機能に依存したプログラムを開発するときには、必要不可欠な資料である。テクニカルデータブックなどからの単なる引き写しではなく、プログラマが必要とする情報の集大成を目指して作成した。

なお、本書は、山崎博義氏、國井淳氏、古庄歩氏、鈴木雅彦氏の多大な協力なしには完成し得なかったことを記しておく。ライブラリのデバッグに関しては鷲北賢氏と西村克信氏にご協力いただいた。また、資料編の一部には日経MIXのdotera氏、ニフティサーブの安井啓祐氏の解析された情報を利用させていただいた。日経MIXのatlan、ma_、mezaemon、shintani、taniyama、

tomina、zom 各氏には、器材や情報の提供でお世話になった。ご協力いただいた方々に深く感謝する次第である。

本書の執筆は、CPU や資料集に関しては小高が、シリアルポートやテキスト VRAM、割り込みに関しては清水が、GDC や GRCG、EGC などグラフィックに関しては速水が担当した。

本書が、より高速で実用的なプログラミングを行おうとしている多くのプログラマーの一助となれば幸いである。

1992 年 2 月 24 日

小高輝真、清水和文、速水　祐

本書の構成

本書は、ひととおりのプログラミングをマスターした方を対象に、ハードウェアを直接操作して PC-9801 の能力を十二分に活かすプログラミングのテクニックを解説しています。

解説にあたっては、それぞれのテーマを操作するハードウェアの種類ごとに分類して、つぎのように各編にまとめてあります。

CPU 編

PC-9800 シリーズで使われている各 CPU の特徴について簡単に説明したあと、それぞれの CPU の判別法、PC-9801 のソフトウェアからのリセット、システムクロック周波数の取得について解説します。

テキスト編

テキスト VRAM を利用する方法や注意点、テキスト画面関係の制御方法などについて説明します。具体的な技法としては、テキスト VRAM を直接操作して文字を表示する方法、カーソルの制御、ユーザ定義文字の登録などについて解説します。

グラフィック基礎編

現在までの PC-9801 をグラフィックのハードウェアで分類し、標準的なグラフィックハードウェアの構成を説明します。そのうえで、グラフィックを使うための初期化の方法を、パレットの操作、VSYNC 信号の検出について解説します。

GDC 編

GDC が持っている機能の概要とコマンドのフォーマットを紹介します。それをもとに、GDC のコマンドを使って図形を高速に描画する方法や、グラフィック画面の 4 プレーンへの書き込み方法、グラフィック画面へ高速に文字を表示する方法について解説します。

GRCG 編

GRCGの3つのモードについて説明したあと、複数プレーン同時書き込みの機能を使った領域の塗り潰しや画面消去について解説します。

EGC 編

EGCが提供する多彩な描画機能について、ラスタオペレーションを中心に説明します。例として、GRCG編でとりあげた塗り潰し四角形の描画をEGCを使って実現する方法について解説します。また、GDC編で取り上げたグラフィック画面への文字表示をEGCを使って実現する方法を解説します。

キーボード編

PC-9800シリーズのキーボードの種類を整理し、それらの判別法について解説します。また、キーボードにコマンドを送信して、CAPSキーやカナキーの状態を操作したり、キーボードの種類を調べたりする方法などを解説します。

シリアルポート編

RS-232C BIOSを使用せずにシリアルポートのハードウェアを直接操作してデータの送受信を行う方法を解説します。また、例として簡単な通信プログラムを作成します。

割り込み編

インターバルタイマやシリアルポート、マウスなどを直接制御するプログラムでは、かならずハードウェア割り込み機能を使います。ここでは、それをコントロールしている割り込みコントローラを操作し、ハードウェア割り込みで処理を行う方法を解説します。

その他周辺機器編

プリンタ、ディスク、マウス、サウンドボードなどの周辺装置を操作するときに必要となる細かいテクニックについて解説します。

本書の読み方

本書の各編は、そのなかで基礎知識と個々のテクニックの解説にわかれています。基礎知識では、各編のテクニックの解説を読むうえで必要になる、各々ハードウェアの全体について解説しています。各編を読むには、まず最初に基礎知識の節を読んでから個々のテクニックの節に移ってください。

基礎知識以外の個々のテクニックの節は、1節ごとに1つのテーマを提示し、その実現方法を説明しています。それぞれの節は、つぎのように構成されています。

■ポイント

その節でどのようなテクニックを用いるかを示します。

■処理の手順

手順をフローチャートで示します。フロー中の項目番号は、プログラミングテクニックの項目番号と対応しています。

■プログラミングテクニック

実際にその節の目的を実現する方法について、「処理の手順」で示した順を追って解説します。

■ワンポイント

その節に関連したトピックや、その節で説明されたテクニックを使うときの注意点などについて述べます。

■ライブラリ

その節で解説されたテクニックを用いて実際に作ったライブラリ関数を紹介します。説明はつぎのようにわかれています。

- **関数名**……………C 言語などから呼ばれる関数名です。
- **解説**…………………関数の説明と、呼び出しの書式を示します。
- **戻り値**……………関数から返される値と意味を説明します。
- **サンプル**…………関数を使ったサンプルプログラム（添付ディスクに含まれる）の名前です。

本書の記述について

本書はつぎのような規則で記述されています。

■文献の表記

文献は『』で囲んで示します。なお、本文中の『テクニカルデータブック』とは『PC-9800シリーズテクニカルデータブック』(アスキー出版局刊)のことを指します。

■数値の表記

*xx*Hは16進数を、*xx*Bは2進数を、それ以外は10進数の数値を表します。

■機種の表記

PC-9800シリーズと表記した場合は、シリーズ全機種を指します。機種名を併記するときには'／'で区切ります。

■項目の参照

「プログラミングテクニック」中の項目を参照する場合には、**1**のように表記します。

■アドレス等の表記

本文中で太字で示した部分は、そこで参照しているメモリアドレスやI/Oポートアドレス、BIOSファンクションなどを表します。

■本文中のプログラムリスト

本書で例示するプログラムリストは、言語環境として、アセンブラではMASM 5.1以上およびTASM 2.0以上を、CコンパイラではQuickC 2.0、MS-C 6.0、Turbo C++2.0、Borland C++ 2.0を想定しています。ただし、本書に掲載したプログラムリストは、添付ディスクに収録したライブラリから必要な部分を切り出したもので、それだけで動作するわけではありません。ご注意ください。

添付ディスクについて

本書には5.25インチおよび3.5インチのフロッピーディスクが添付されています。ディスクにはつぎのものが含まれています。

- ●本書で説明した全ライブラリのソースコードと関連ファイル
- ●C言語から使えるライブラリの.LIBファイル
- ●ライブラリの関数を実際に使ったサンプルプログラム

添付したライブラリは基本的にノーマルモードのPC-9800シリーズ全機種に対応しています。一部、動作しない機種のある関数もありますが、それについては本文中で適宜説明し、また各関数の動作条件をAppendixの「ライブラリ関数の動作条件」に掲載しました。

ディスクの内容と実際の使用法については、Appendixをお読みください。

プログラム利用上の注意

本書に掲載されたプログラムやディスクに収録したプログラムの多くは、ハードウェアを直接操作しており、その性格上デバイスドライバや常駐ソフトなどの他のソフトウェアとの競合によりプログラムの正常な動作が行なわれない場合や、他のソフトウェアの動作に対して悪影響を与える場合があります。プログラムの実行に当たっては、本文を参照しその挙動を十分ご理解の上、ご利用ください。

なお、このディスクに収録されたプログラムは次のような条件で利用できます。

- プログラムを私的利用の範囲において使用すること
- プログラムを私的利用の範囲において、内容を変更、翻案および複製すること

また、以下のような事項を禁止します。

- 本ディスクに収められたプログラムの全部または一部を、複製し頒布すること
- プログラムに表示されている著作権その他権利者の表示を削除したり変更を加えること

本書に掲載されたプログラムの運用結果については一切責任を負いかねますのでご了承ください。

目　次

CPU編

ＣＰＵ編

CPU の基礎知識

PC-9800 シリーズは全機種とも 80x86 系の CPU を搭載している。80x86 系 CPU は新世代のものほど命令が拡張され、実行速度も向上している。また、途中からプロテクトモードや仮想記憶をサポートするなど、CPU 自体のアーキテクチャも変更されている。

ここでは、リアルモードでのプログラミングに関係する各 CPU の特徴について簡単に解説する。プロテクトモードの動作や各命令の詳細はそれぞれの CPU のデータブックに譲る。

PC-9800 シリーズの CPU の種類

PC-9800 シリーズに採用されている CPU の一覧を Table 1 に示す。機種別の搭載 CPU、CPU クロック一覧は、『テクニカルデータブック』の「機種別仕様」を参照のこと。

Table 1　PC-9800 シリーズが採用している CPU の一覧

通　称	デバイス名	コプロセッサ名	搭載機種
8086	i8086（インテル）	8087	PC-9801 初代/E/F/M
80286	i80286（インテル）	80287	PC-9801VX、PC-98XA など
80386DX	i80386DX（インテル）	80387DX	PC-9801RA など
80386SX	i80386SX（インテル）	80387SX	PC-9801ES など
80486DX	i80486DX（インテル）	CPU に内蔵	PC-H98
80486SX	i80486SX（インテル）	80487SX	PC-H98S など
V30	μPD70116（NEC）	8087	PC-9801U2/VM など
V33	μPD70136（NEC）	PC-98DO +には搭載不可	PC-98DO +
V50	μPD70216（NEC）	PC-98LT/HA には搭載不可	PC-98LT/HA

8086

80x86 系 CPU のベースとなっているのが最初にリリースされた 8086 である。8086 の命令セットはすべての 80x86 系 CPU が備えているので、その範囲で記述すれば未定義命令（POP CS、MOV CS,XX など）を使用していない限りどの 80x86 系 CPU でも実行できる。つまり、全機種対応のプログラムを作るときには 8086 命令セットで記述しなければならない。

80186

PC-9800シリーズでは採用されていないが、80186は80x86系のなかで1つの区切りとなる命令セットを持っているため取り上げる。

基本的な命令の拡張や新命令の追加などが行われているので80186命令セットの利用価値は高い。複数レジスタのPUSH命令やPOP命令、2〜3オペランドの符号付き乗算、即値のPUSH、1以外の即値のローテートやシフトなどはアセンブラのプログラミングでは便利だろう。

80186の直系に当たる80286、80386、80486はもとより、NECのVシリーズCPUも80186命令セットを踏襲している。したがって、PC-9800シリーズではPC-9801初代/E/F/M以外の機種ならば80186命令セットを使用できる。ただし、V30やV50は80186がサポートしている無効命令の検出機能を持っていない。V33は無効命令の検出機能は持っているが、割り込み番号が80186とは異なりINT 7AHである。

NEC Vシリーズ

V30はPC-9800シリーズのなかでもっとも多くの機種に搭載されているCPUである。また、V50がPC-98LT/HAに、V33AがPC-98DO+に採用されている。

V30では80186命令セットにいくつかの独自命令が追加されている。また、一部の命令の実行速度は80186より高速になっている。ただし、V30独自命令はV33、V50などVシリーズCPU以外はサポートしていないことに注意しなければならない。MASM系のアセンブラもV30独自命令はサポートしていない。

V50はV30をコアにペリフェラルデバイスを1チップに集積したCPUで、命令セットはV30とコンパチブルである。

V33はV30のアーキテクチャを基本にしているが、命令をワイヤードロジックで実行することで処理速度を大幅に向上させたCPUである。メモリ空間は16Mバイトまで取り扱えるが、80286のような保護機能は内蔵していない。命令セットには拡張されたメモリ空間の制御用命令が追加されている。なお、V33はPUSH SPでスタックにプッシュされる値が80286以降と同じである。CPUが80286以降かどうかをこの方法で判断するとV33のときに誤認するので注意しなければならない。

80286

CPUのメモリ空間は8086では最大1Mバイトだったが、80286では16Mバイトまで扱うことができるように拡張されている。また、保護機能を備えたプロテクトモードもサポートしている。

命令の実行速度は8086に比べて2倍程度向上している。命令セットの拡張はプロテクトモード

関係だけなので、一般的なMS-DOS上のプログラム開発をする限りは80186命令セットと同一と考えてよい。

なお、プロテクトモードを利用するプログラムを作るときには、CPUアーキテクチャのほかにPC-9801固有の事柄についても理解しておかなければならない。プロテクトメモリにアクセスするにはアドレスバスのbit 20以上をアクティブにするI/O命令を実行する必要がある。また80286にはプロテクトモードからリアルモードに戻る命令が存在しないので、リアルモードに戻るにはI/OポートからCPUをリセットしなければならない。これらの仕組みはCPU外部のハードウェアで実現されているので、PC-9801固有の方法を用いる必要がある。同じ80286を搭載していても、その他のアーキテクチャの機種（IBM-PCなど）では当然その方法も異なっている。

80386

80386ではCPUのアーキテクチャが16ビットから32ビットへと変化した。これにともなってレジスタも32ビット幅になり、アドレス空間も4Gバイトまで拡張された。また、ページング、仮想86モードなどの進んだ機能も取り入れられている。

基本命令ではビット操作命令をはじめとする新命令が追加された。80286の欠点とされていたプロテクトモードからリアルモードへの移行もソフトウェアで可能になった。

実行速度に関しては、データバス幅が32ビットになった（80286までの倍）ため、32ビット対応命令でブロック転送を実行すると従来の2倍の速度で転送できるようになった。これはリアルモードでも可能である。ただし、転送開始アドレスが4で割り切れる値(4バイト＝32ビット)でなければ最大のパフォーマンスは発揮できない。このほかにも一部の命令が80286より少ないクロック数で実行できるようになっているが、機能が複雑になった分I/O命令などを中心に実行クロック数が増加していることも知っておいたほうがよいだろう。

80386DXと80386SX/SL (98) の相違点はCPU外部に引き出されているバス幅の違いだけで、ソフトウェア的にはほぼ完全な互換性がある。DXがデータバス幅32ビット、アドレスバス幅32ビット(4Gバイト)なのに対し、SX/SL(98)はデータバス幅16ビット、アドレスバス幅24ビット（16Mバイト）である。80386SXでデータを32ビット転送したとき、理論上は16ビットで転送したときと同速度になるはずであるが、PC-9801ESで実測したところ32ビット転送は16ビット転送の80%程度の時間で終了した。これはメモリのアクセス順序の違いが速度の違いにつながるメモリシステムの特性によるものと思われる。同様に、80386DXで拡張スロットのメモリ（16ビット幅）間で転送を行ったときにも高速化が認められた（その度合はメモリボードの速度による）。

80486

80486には80486DXと80486SXの2種類がある。80486DXは、1チップ上に80386互換のCPUコア、コプロセッサ、8Kバイトのキャッシュを集積したCPUである。CPUコアのソフトウェア的なアーキテクチャは80386とほぼ同一であるが、基本命令をワイヤードロジックで実行するように設計されているため(複雑な命令は従来どおりマイクロコードで実行する)、命令の実行速度は非常に高速になっている。単純なレジスタ間演算ならわずか1クロックで終了してしまう。ごく一部の命令で80386より多くの実行クロックを費やすものもあるが、問題にはならないだろう。いくつかの新命令の追加もあるが、キャッシュコントロール用命令やマルチプロセッサ対応命令なので、特別なプログラム以外では利用することはない。

80486SXは80486DXからコプロセッサ機能をなくしたものだが、外部にコプロセッサのみを増設することはできない。コプロセッサが必要な場合には80487SXを増設する。すると、元から実装されていた80486SXは動作を停止して、80487SXがCPUとコプロセッサの機能を代行するようになる。つまり、80487SXは80486DXとまったく同じ機能を持っているということである。

仮想86モード（V86モード）

仮想86モードは80386、80486のモードの1つでCPUの名称ではない。しかし、仮想86モードでは80386、80486の機能の一部が制限される（たとえばさらに仮想86モードに入ることはできない）状態になるため、CPUの一つの動作モードとしてここで取り上げる。

仮想86(Virtual 86またはV86)モードは仮想マシンを実現するために用意されている機能で、実際にはプロテクトモードの一形態である。このため、I/Oやメモリアクセスの保護機能なども備わっているが、MS-DOS上ではもっぱらEMM386のようにソフトウェアで実現するEMSドライバなどに利用されている。

CPUが仮想86モードのときは、さらにプロテクトモードへ移行したり、プロテクトモードの設定に必要なレジスタを書き換えたりすることはできない。そのため、そのような動作をするプログラムではCPU判定だけでなく、仮想86モードかどうかも判別する必要がある。

各CPUの比較

以上、各CPUの特徴について説明した。各CPUの継承関係を図に表すと、Figure 1のようになる。

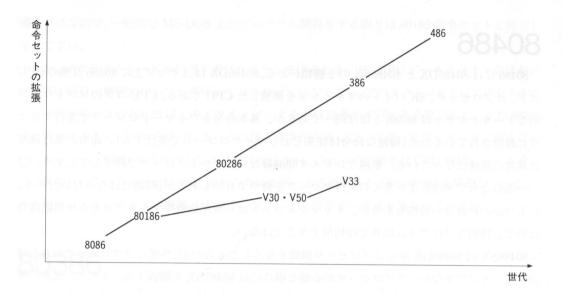



Figure 1　命令セットの継承

また、各CPUでの命令の拡張をTable 2に示す。

Table 2　命令セットの拡張一覧

CPU	追加命令	拡張命令	備考
80186	BOUND、ENTER、LEAVE、INS、OUTS、PUSHA、POPA	IMUL、PUSH、RCR、RCL、ROR、ROL、SAR、SAL、SHR、SHL	配列境界チェック、スタックフレーム生成・削除、ブロックI/Oリード・ライト、複数レジスタのPUSH・POP命令が追加された。 2・3オペランドの符号付き乗算、即値のPUSH、1以外の即値のローテート・シフトが可能なように命令が拡張された。
V30、V33、V50	INS、EXT、ADD4S、SUB4S、CMP4S、ROL4、ROR4、TEST1、NOT1、CLR1、SET1、BRKEM、FPO2		ここの表記のみNECニーモニック。INSは80186のINSとは異なる命令。 V33はBRKEM（8080Aエミュレーション）をサポートしていない。 ビット操作命令、BCD演算命令などが追加された。ただし、これらの命令はVシリーズCPUのみサポートしている。 80186の命令セットはすべて含んでいる。ただし、V30とV50には無効命令検出がない。V33は無効命令検出を行うが、割り込み番号が異なる（INT 7AH）。
80286	SGDT、LGDT、SIDT、LIDT、CLTS、LMSW、SMSW、VERR、VERW、STR、		プロテクトモード制御用命令が追加された。リアルモードのみで使用する場合には80186の命令セットと同等である。

CPU	追加命令	拡張命令	備　考
	SLDT、LLDT、LTR、LSL、LAR、ARPL		
80386	BSF、BSR、BT、BTC、BTR、BTS、CDQ、CWDE、SHLD、SHRD、MOVSX、MOVZX、SET*xx*、LFS、LGS、LSS	CMPS、SCAS、LODS、STOS、MOVS、MOV、IMUL、PUSHA、POPA、PUSHF、OPF、J*xx*、LOOP、IRET、INS、OUTS	多彩なビット操作命令、符号拡張命令、条件処理命令などが追加された。アドレス操作をともなう命令が32ビット対応に拡張された。80386アーキテクチャを制御できるように一部の命令が拡張された。80286の命令セットはすべて含んでいる。
80486	INVD、WBINVD、INVLPG、BSWAP、CMPXCHG、XADD		内蔵キャッシュの制御命令、バイトオーダーの入れ替え命令等のマルチプロセッサ対応命令などが追加された。80386の命令セットはすべて含んでいる。

各CPUがサポートするエクセプション(例外)はTable 3のとおりである。

Table 3　CPU別サポートエクセプション一覧

番　号	例　外	サポートするCPU
INT 00H	除算エラー	8086～、V30、V50、V33
INT 01H	シングルステップ	8086～、V30、V50、V33
INT 02H	NMI	8086～、V30、V50、V33
INT 03H	ブレークポイント	8086～、V30、V50、V33
INT 04H	INTO オーバーフロー検出	8086～、V30、V50、V33
INT 05H	BOUND 境界チェック	80186～、V30、V50、V33
INT 06H	無効命令	80186～
INT 07H	コプロセッサ不在	80286～
INT 08H	ダブルフォールト	80286～
INT 09H	コプロセッサセグメントオーバラン	80287～（80486は除く）
INT 0AH	無効タスクステートセグメント	80286～
INT 0BH	不在セグメント	80286～
INT 0CH	スタック・フォールト	80286～
INT 0DH	一般保護フォールト	80286～
INT 0EH	ページフォールト	80386～
INT 0FH		
INT 10H	コプロセッサ・エラー	80287～
INT 11H	アラインメント・チェック	80486～

CPUの判別

8086より上位のCPUに備わっている機能（命令など）を使うときには、あらかじめCPUの種類を判別して、その機能が使えることを確認しておく必要がある。

CPUの種類を判別する方法には、大きく分けて2とおりある。システム共通域を参照する方法と、CPUによってふるまいの異なる命令を実行させて、その結果から判断する方法である。

システム共通域を参照する方法では、本来その機種に搭載されているCPUの種類を知ることができる。しかし、今後発表される機種でシステム共通域のCPU種別の表現方法が変更される可能性も考えられる。

一方、CPUによってふるまいの異なる命令を実行させてその結果から判断する方法は、アドオンボードでCPUを交換したような場合にも正確にCPUの種類を知ることができる。

ここでは、後者の方法を用いたCPUの判別ルーチンについて解説する。

▶ポイント

- 80x86系のCPUには、CPUの種類を知るための機能（命令等）は用意されていない。
- 実行するCPUによって挙動が変わる命令があるので、これを利用してCPUを判別する。
- 挙動の変わる命令は複数あり、さまざまな判別法が考えられる。
- 一回の判定で、CPUを2つに分類できる。どう分類されるかは判定方法によって異なるので、特定のCPUを判別するには複数回の判定を行う必要がある。
- 80386以上のCPUでは、仮想86モードの場合もあるので必要ならばこれも判別する。

▶プログラミングテクニック

■CPUの判別の方針

ポイントで述べたように、CPUを判別する方法は複数考えられるが、ここでは、つぎに紹介する8つの判別法をとりあげる。なお、番号は各判別法を説明している項を示す。

❶ シフト・ローテート命令の動作の違いを利用した判定
❷ AAD命令の隠し機能を利用した判定
❸ 0FHコードを利用した判定
❹ PUSH SPの動作を利用した判定
❺ フラグレジスタ（bit15～12）の動作の違いを利用した判定

(1)bit 15が常に0かつbit 14の書き換えが可能かの判定
(2)bit 15が常に0かの判定
(3)bit 14が常に0かの判定
(4)bit 13～12の書き換えが可能かの判定

❻ MSWを利用した仮想86モード判定
❼ ACフラグの有無を利用した判定
❽ 無効命令例外を発生させる判定

これらはTable 1のようにCPUをさまざまに類別する。ほとんどは、AグループとBグループの2つに判別できる。また判別法によってはA～Dまでの4つに判別できる。×となっているところは実行してはならない。

Table 1　判別法と結果

	❶	❷	❸	❹	❺ (1)	❺ (2)	❺ (3)	❺ (4)	❻	❼	❽
8086	A	B	B	A	B	B	B	B	×	×	×
80186	B	B	×	A	B	B	B	B	×	×	A
80286	B	B	×	B	B	A	A	B	A	×	B
80386（リアルモード）	B	B	×	B	A	A	B	A	A	A	C
80386（仮想86モード）	B	B	×	B	A	A	B	B	B	A	C
80486（リアルモード）	B	B	×	B	A	A	B	A	A	B	D
80486（仮想86モード）	B	B	×	B	A	A	B	B	B	B	D
V30/V50	A	A	A	A	B	B	B	B	×	×	×
V33	A	B	A	B	B	B	B	B	×	×	×

実際には、これらを組み合わせて特定のCPUを判別したり、利用したい命令が実行できるかどうか判定する。たとえば、Vシリーズ特有命令が使えることを確認したいときは、❶と❸の方法を組み合わせれば良いことがわかるだろう。

■1 シフト・ローテート命令の動作の違いを利用した判別法

この方法では、8086、V30、V33、V50 と 80186、80286、80386、80486 を判別できる。

8086 と V シリーズ CPU は、シフト・ローテート命令を実行するとき CL レジスタで指定された回数分だけ実際にシフトを行う。それに対して 80186 以降の 80x86 ではシフト回数の最大値は 31 であり、CL は下位 5 ビット（bit4〜0）を残してマスクされる。

プログラムを List 1 に示す。

List 1　シフト・ローテート命令の動作の違いを利用した判定法

```
;--- 8086、V30、V33、V50 ／ 80186、80286、80386、80486 判別
;       AL ← 01H、 CL ← 20H とセットして、SHR AL,CL を実行する。
;       8086 と V シリーズでは 20H (32) 回シフトが行われて AL=0 となる。
;       80186 以上の CPU では 20H and 1FH=00 となり、シフトは行われず AL=1 となる。
                mov     al,01H                          ; 被シフト値
                mov     cl,20H                          ; シフト回数 (32 回)
                shr     al,cl
                jz      (8086,V30,V33,V50)              ; シフトが行われて AL=0
                jmp     (80186,80286,80386,80486)       ; シフトは行われず AL=1
```

■2 AAD 命令の隠し機能を利用した判別法

この方法では、V30、V50 と 8086、80186、80286、80386、80486、V33 が判別できる。

AAD 命令はアンパックド BCD 数 (2 進化 10 進数) を通常の 2 進数に変換する命令である。たとえば、10 進数の 28 を表すアンパックド BCD 数は 0208H であるが、これを 2 進数に変換するには 2×0AH＋8＝1CH という演算を行えばよい。AAD 命令は AX レジスタにセットされた値に対してこの演算を行い、AL レジスタに格納する。AH レジスタは 0 になる。

さて、実際の AAD 命令のオペコードは D5H, 0AH という 2 バイト命令である。実はこの 2 バイト目の 0AH という値は、80x86 では被変換数の 10 の位 (AH) に対する乗数なのである。ところが、V30 や V50 では 2 バイト目の値に関らず、常に 0AH を乗ずる。この違いを利用して V30 や V50 とその他の CPU を区別することができる。なお、V33 は 80x86 と同様の動作をした。

2 バイト目が 0AH 以外の値をとる AAD 命令は MASM では記述できないため、DB 命令を使って記述する。なお、Turbo Assembler では AAD 00 のような記述が可能である。

プログラム例を List 2 に示す。

List 2　AAD命令の隠し機能を利用した判定法

```
;--- V30・V50／8086・80186・80286・80386・80486・V33 判別
;       AX ← 0100H をセットして、AAD 00 を実行する。
;       V シリーズ（除く V33）では AL が 01*0AH+00=0AH となる。
;       80x86・V33 では AL が 01*00H+00=00H となる。
                mov       ax,0100H              ; 被変換値
        DB      0D5H,00H
        ;80x86  : AAD 00         ;AL=AH*00H+AL, AH=00
        ;V30    : AAD             ;AL=AH*0AH+AL, AH=00
                jz        (8086,80186,80286,80386,80486,V33)   ;0 が乗算されて AL=0
                jmp       (V30,V50)                            ;10 が乗算されて AL=0AH
```

3 0FHコードを利用した判別法

この方法では、V30、V33、V50と8086が判別できる。

0FHコードは8086ではPOP CSの動作をする。オペコードマップを見れば、〝PUSH DS〟(1EH)、〝PUSH CS〟(0EH)、〝POP DS〟(1FH)であることから〝POP CS〟(0FH)と容易に推定できるため、良く知られた隠し命令である。

同様の隠し命令に〝MOV CS,AX〟(8E C8H)がある。〝MOV AX,DS〟(8C D8H)、〝MOV AX,CS〟(8C C8H)、〝MOV DS,AX〟(8E D8H)から推定できる。8086、V30、V50、V33で実行できるが、80286以上のCPUでは無効命令例外が発生する。

0FHはVシリーズCPUでは固有の拡張命令のプリフィックスとして使われている。80286以上のCPUでもVシリーズとは異なる拡張命令プリフィックスとして使用されている。80186では0FHコードを実行すると無効命令例外が発生する。

この0FHコードの動作の違いを利用してVシリーズと8086を判別することができる。ただし、多少の工夫が必要になる。8086では0FHは1バイト命令であるが、Vシリーズでは複数バイト命令の1バイト目である。判別を行うには、どちらのCPUでも実行できるコード列であって、さらにCPUによって実行結果が変化するプログラムでなければならない。List 3に挙げるプログラムは8086とV30の命令コード表を見比べながら妥当と思われる組み合わせを見つけだして作成した。

0FH、14H、C0Hというコード列は、V30と8086でList 4のように解釈される。このコード列を実行する前にあらかじめSTC命令でCF=1としておけば、V30ならCFは変化しないのでCF=1、8086ならADC命令でCFがクリアされてCF=0となって判別できる。

ところで、8086のときにはPOP CSが実行されるのでまえもってPUSH CSしておく必要がある。そうしておかなければ、POP CSしたときにCSに予期せぬ値が入り制御がまったく異った所に行ってしまうからだ。逆に、V30のときにはPOP CSは実行されないのであとでスタックポインタを元に戻しておかなければならない。このため、最初にスタックポインタの位置を保存し、最後にそれを元に戻している。

List 3 0FHコードを利用した判定法

```
;--- V30、V33、V50／8086判別
;       8086、V30、V33、V50以外のCPUでこのルーチンを実行させてはならない
                mov     bx,sp           ;スタックポインタ保存
                push    cs              ;POP CSに備えてCSをPUSH
                mov     al,0            ;adc al,0C0Hで繰り上がらないように
                stc                     ;CF=1
        DB      0FH,14H,0C0H
        ;V30  : set1    al,cl           ;ALのbit clがsetされる,CF=no change
        ;8086 : pop     cs
        ;8086 : adc     al,0C0H         ;AL=0C1H、CF=0
                mov     sp,bx           ;スタックポインタ復帰
                jc      (V30,V33,V50)
                jmp     (8086)
```

List 4 0FH、14H、C0HのV30と8086での解釈の違い

```
;V30の場合
;ALのCLで指定されたビットが1になる。CFは変化なし
        set1    al,cl

;8086の場合
        pop     cs
        adc     al,0C0H
;ALにC0HとCFが加算される。CFは0になる
```

4 PUSH SPの動作を利用した判別法

この方法では、8086、80186、V30、V50とV33、80286、80386、80486が判別できる。

80286以降の80x86とV33では、PUSH SPのとき命令実行前の値がPUSHされる。その他のCPUではSPを減じたあとの値がPUSHされる。

V33が80286以上のCPUと同じ動作をすることに注意する必要がある。プロテクトモード命令が使えるかどうかをこの方法で判定するとV33のとき異常動作することになる。

プログラム例をList 5に示す。

List 5　PUSH SP の動作を利用した判定法

```
;--- 8086、80186、V30、V50 ／ V33、80286、80386、80486 判別
                push    sp
                pop     ax
                cmp     ax,sp
                je      (V33,80286,80386,80486)
                jmp     (8086,80186,V30,V50)
```

5 フラグレジスタ(bit15〜12)の動作の違いを利用した判別法

フラグレジスタの上位4ビットはCPUによってTable 2のように異なったふるまいをする。

Table 2　フラグレジスタ

	bit15	bit14	bit13	bit12
8086、80186、Vシリーズ	1	1	1	1
80286	0	0	0	0
80386、80486 リアルモード	0	×	×	×
80386、80486 仮想 86 モード	0	×	1/0	1/0

フラグのbit14(NT)はプロテクトモード時のIRET命令の動作を指定するためのもので、80286のリアルモードではここは常に0を示す。80386、80486のリアルモードと仮想86モードでは書き換え可能であるが、ここを書き換えてもプロテクトモード以外ではIRET命令の動作に影響を与えることはない。

bit13〜12(IOPL)はI/O特権レベルを指定するためのもので、80286のリアルモードではここは常に00Bを示す。80386、80486のリアルモードでは書き換え可能であるが、ここを書き換えてもCPUの動作に影響を与えることはない。仮想86モードでは書き換え不可能になる。ほとんどの仮想86モニタは11Bに設定しているが、仮想86モニタの設計によってはこれ以外の値をとることがある。

以上のようなフラグの特性を利用してCPU判別を行うプログラムのいくつかをList 6、List 7、List 8、List 9に示す。

List 6　bit 15 が常に 0 かつ bit 14 の書き換えが可能かの判定

```
;--- 80386・80486 ／ 8086・80186・80286・V30・V33・V50 判別
;       80386 命令を使用するときにこの判定が利用できる。
;       80386・80486 では bit 15 は常に 0、bit 14 は書き換え可能なので、
;       bit 14=1 を書き込んだとき、bit 15,14=0,1 が読み出せれば
;       80386・80486 である。
                pushf
                pop     ax                      ;mov ax,FLAGS
                pushf                           ;FLAGS 待避
                or      ax,4000H                ;FLAGS bit 14=1 をセット
                push    ax
                popf                            ;mov FLAGS,ax
                pushf
                pop     ax                      ;mov ax,FLAGS
                popf                            ; フラグ復帰
                and     ax,0C000H               ;FLAGS bit 15,14 を残してマスク
                cmp     ax,4000H                ;FLAGS bit 15,14=01 なら 80386 以上
                je      (80386,80486)
                jmp     (8086,80186,80286,V30,V33,V50)
```

List 7　bit 15 が常に 0 かの判定

```
;--- 80286・80386・80486 ／ 8086・80186・V30・V33・V50 判別
;       80286・80386・80486 では bit 15 は常に 0 なので、bit 15=0 が
;       読み出せれば 80286・80386・80486 である。
                pushf                           ;mov ax,FLAGS
                pop     ax
                test    ax,8000H                ;FLAGS bit 15=0 なら 80286 以上
                jz      (80286,80386,80486)
                jmp     (8086,80186,V30,V33,V50)
```

List 8　bit 14 が常に 0 かの判定

```
;--- 80286 ／ 8086・80186・V30・V33・V50・80386・80486 判別
;       80286 では bit14 は常に 0 なので、bit 14=1 を書き込んで、
;       0 が読み出せれば 80286 である。
                pushf
                pop     ax                      ;mov ax,FLAGS
                pushf                           ; フラグ待避
                or      ax,4000H                ;FLAGS bit 14=1 をセット
                push    ax
                popf                            ;mov FLAGS,ax
                pushf
                pop     ax                      ;mov ax,FLAGS
                popf                            ; フラグ復帰
```

```
            test    ax,4000H        ;FLAGS bit 14=0 なら 80286
            jz      (80286)
            jmp     (8086,80186,V30,V33,V50,80386,80486)
```

List 9　bit 13〜12 の書き換えが可能かの判定

```
;--- 80386・80486 Real Mode／
;       80386・80486 V86 Mode・8086・80186・80286・V30・V50・V33 判別
;       80386,80486 のリアルモードでしか実行できない機能（CRn レジスタ等の書き換え、
;       V86 モードへの移行など）を使用するときにこの判定が利用できる。
;       80386,80486 のリアルモードのみ bit 13〜12 が書き換えできるので、ここを反転
;       させた値を FLAGS に書き込み、それを読みだして元の FLAGS と異なれば 80386,80486
;       リアルモードと判断できる。
            pushf
            pop     ax              ;mov ax,FLAGS
            pushf                   ; フラグ待避
            mov     bx,ax           ; 元の FLAGS の値を保存
            xor     ax,3000H        ;FLAGS bit 13〜12=を反転
            push    ax
            popf                    ;mov FLAGS,ax
            pushf
            pop     ax              ;mov ax,FLAGS
            popf                    ; フラグ復帰
            cmp     ax,bx           ; 元の FLAGS の値と比較
            jne     (80386,80486 Real Mode)
            jmp     (80386,80486 V86 Mode, 80286,80186,8086,V series)
```

6 MSW を利用した仮想 86 モード判別法

この方法では、80386、80486 のリアルモードと仮想 86 モードが判別できる。

MSW（Machine Status Word）の bit 0（プロテクトモードフラグ）を調べると仮想 86 モードかどうかを知ることができる。ここが 1 ならば仮想 86 モードである。80286 以上の CPU で実行できるが、当然のことながら 80286 では常にリアルモードを示す。

なお、EFLAGS の bit 17 には VM（Virtual 8086 Mode）フラグがあるが、ここを読みだすと仮想 86 モードでも常に 0 を示すので、現在仮想 86 モードかどうかを知るためには使えない。

プログラム例を List 10 に示す。

List 10 MSW を利用した仮想 86 モード判定法

```
;--- Real Mode ／ V86 Mode 判別
;       80286・80386・80486 のみで実行すること。
;       8086・80186・V30・V33・V50 で実行させてはならない。
        .286P
                smsw    ax          ;mov ax,MSW(CR0)
                test    al,1        ;MSW の bit 0(Protect Mode Flag) をチェック
                jz      (Real Mode)         ;PMF=0
                jmp     (V86 Mode)          ;PMF=1
```

7 AC フラグの有無を利用した判別法

この方法では、80386 と 80486 が判別できる。

80486 では EFLAGS のビット 18 に AC（Alignment Check）フラグが割り当てられている。アラインメントチェックとは、32 ビット境界をまたいだメモリアクセスをしたときに AC 例外(INT 11H）を発生させる機能である。AC フラグと CR0 レジスタ内の AM（Alignment Mask）ビット（bit 18）が共に 1 で、CPL（Current Privilege Level）が 3 のときアラインメントチェックが有効になる。80486 では AC ビットを 1 に書き換えることができるが、80386 では常に 0 である。これを利用して 80386 と 80486 を判別することができる。

EFLAGS を変更するには POPFD 命令を使ってスタック経由で値を書き込まなければならないが、AC ビットを 1 にしたとき、もしスタックポインタ (SP) が 32 ビット境界に揃っていなかったとすると AC 例外が発生してしまう可能性がある。これを防ぐためにあらかじめ SP を 32 ビット境界に調整しておかなければならない。調整は、FFFCH と AND をとる(つまり下位 2 ビットをクリアする) だけで良い。SP が 32 ビット境界をまたぐ値 (4 で割り切れない値) を持っていた場合には、最も近い 4 で割り切れる値に減算される。この操作はスタックが消費されたのと同じなので何の問題も生じない。AC ビットのチェックの後で SP を元の値に戻しておけば良い。

また、AC が 1 の期間は割り込みの禁止も必要である。割り込みルーチンのなかで 32 ビット境界をまたぐメモリアクセスが起り得るからだ。

80386 命令セットを使用する前には、.386 ディレクティブでアセンブラにその宣言を行う必要がある。また、80386、80486 以外の CPU が実行する部分では.8086 ディレクティブなどで 32 ビット命令の使用を抑止しておくほうが良い。なぜなら、明示的に 32 ビット命令を記述しなかったとしても、条件ジャンプの部分で 80386 以降でのみサポートしている 16 ビット相対条件ジャンプ命令が生成されてしまう可能性があるからである。

プログラム例を List 11 に示す。

List 11　ACフラグの有無を利用した判定法

```
;--- 80386／80486判別
;       80386・80486以外で実行させてはならない。
        .386
                mov     bx,sp               ;SP保存
                and     sp,0FFFCH           ;ACが起動しないようSPを32bit境界に調整
                pushfd                      ;EFLAGS待避
                cli                         ;割り込み禁止
                pushfd
                pop     eax                 ;mov eax,EFLAGS
                or      eax,00040000H       ;bit18=1(Alignment Check ON)
                push    eax
                popfd                       ;mov EFLAGS,eax
                pushfd
                pop     eax                 ;mov eax,EFLAGS
                popfd                       ;EFLAGS復帰
                mov     sp,bx               ;SP復帰
                test    eax,00040000H       ;EFLAGSのbit 18をチェック
        .8086   ;16bit相対条件ジャンプ命令が生成されないように.386を解除
                jnz     (80486)             ;bit 18=1
                jmp     (80386)             ;bit 18=0
```

8 無効命令例外を発生させる判別法

80186以降の80x86CPUでは未定義命令を実行したときに無効命令例外（INT 06H）が発生する。これを利用して、どの命令をサポートしているかでCPUを区別することもできる。

しかし、この方法にはMS-Windowsの386エンハンストモードで動作させると特権違反でプログラムが強制的に停止させられてしまうという問題がある。

プログラム例をList 12に示す。

List 12　無効命令例外を発生させる判定法

```
;8086・V30・V50・V33で実行しないこと。
;注）80486の命令を使っている部分があるため、
;       MASM 6.00以降またはTASM 2.0以降でなければアセンブルできない。
                push    ds
                push    es
        ;--- INT 06Hベクタ保存
                mov     ax,3506H
                int     21H                 ;ES:BX ← INT 06H entry
        ;--- INT 06Hベクタ変更
                mov     ax,cs
                mov     ds,ax
```

```
        mov     dx,OFFSET InvCodeDetect
        mov     ax,2506H
        int     21H
        mov     cx,sp                   ;SP保存
;--- 80286特有命令のテスト
        mov     dx,01H
.286P
        smsw    ax                      ;80186では未定義
;--- 80386特有命令のテスト
        inc     dx
.386
        bt      ax,1                    ;80286では未定義
;--- 80486特有命令のテスト
        inc     dx
.486
        bswap   eax                     ;80386では未定義
.8086
        inc     dx
InvCodeDetect:  ;無効命令が検出された場合のジャンプ先
        mov     sp,cx                   ;SP復帰
;--- INT 06Hベクタ復帰
        push    dx
        mov     ax,es
        mov     ds,ax
        mov     dx,bx
        mov     ax,2506H
        int     21H
        pop     dx
;--- レジスタ復帰
        pop     es
        pop     ds
;DX=1 : 80186
;DX=2 : 80286
;DX=3 : 80386
;DX=4 : 80486
```

ライブラリ

CpuKind

解説 CPUの種別を返す。書式はつぎのとおり。

int **CpuKind**(void)

戻り値 各ビットに使える機能の有無を反映して返す。

値	CPU
0	8086
1	80186
2	80286
3	80386
4	80486
11	V30、V50
31	V33
83	80386（V86）
84	80486（V86）

戻り値のビットごとの意味は以下のとおり。

ビット	値	意　味
bit 7	1	仮想86
	0	非仮想86
bit 5	1	V33命令サポートあり
	0	V33命令サポートなし
bit 4	1	V30命令サポートあり
	0	V30命令サポートなし
bit 2〜0	100B	80486命令サポートあり
	011	80386命令サポートあり
	010	80286命令サポートあり
	001	80186命令サポートあり
	000	8086命令サポートあり

サンプル CPUTEST.C

システムクロック周波数の取得

プログラムを作成するときにCPUのクロック周波数を知らなければならないことはあまりない。CPU速度に過度に依存するようなプログラミングは、将来への互換性のためにできるだけ避けたほうがよいからだ。

一方、システムクロックの周波数を取得することは重要である。タイマの割り込み周期やビープの音程、RS-232Cのビットレートを設定するときに、入力周波数から算出した値を8253Aのレジスタにセットする必要があるからだ。ここでは、そのシステムクロック周波数を取得する方法を紹介する。

▶ポイント

- タイマを直接操作するプログラムでは、CPUの動作周波数(いわゆる16MHzや20MHzとカタログに記載されているもの)よりも、システムクロックが何MHzかの方が重要である。
- タイマには、このシステムクロックが入力されており、シリアルポートの通信速度などはこのクロックに依存する。
- システムクロック周波数は、システム共通域0000:0501Hのbit 7に示されている。

▶プログラミングテクニック

1 システム共通域からシステムクロック周波数を取得する

プログラムを作成するに当たって、80386の20MHzや16MHzといったCPUの動作周波数を知る必要はあまりない。一方、システムクロックの周波数を取得することは非常に重要である。システムクロックは8253A(プログラマブル・インターバルタイマ)のクロック源として使用されるので、タイマの割り込み周期などを設定するときに、入力周波数から算出した値を8253Aのレジスタにセットする必要があるからだ。

システムクロックとは、拡張バスのSCLK1端子から出力されている信号で、ほとんどの機種ではCPUクロックの設定と連動している。8253Aにはシステムクロックを2〜4分周した約2.5MHz

または約2.0MHzのクロックが入力されている。Table 1とFigure 1にCPUクロックとシステムクロック、8253Aクロックの関係とシステムクロック周りの回路のイメージを示した。なお、この概念図は実際の回路の接続を表したものではない。例えば、実際の80286、80386にはCPUのクロックとして2倍の周波数が入力されているなど、実際とは食い違う部分もある。

Table 1　CPUクロックとシステムクロック、8253Aクロックの対応

CPUクロック	システムクロック	8253Aクロック
5MHz	4.9152MHz（約5MHz）	2.4576MHz（約2.5MHz）
10、12、20MHz（除PC-H98）	9.8304MHz（約10MHz）	2.4576MHz（約2.5MHz）
8、16MHz、PC-H98	7.9872MHz（約8MHz）	1.9968MHz（約2.0MHz）

システムクロックが半端な数値になっているのは、シリアルポートの通信速度を基準に設定されているからである。たとえば、9600bpsで通信する場合には9600Hzの16倍の周波数が必要になるのだが、9600Hz×16＝153.6kHzの整数倍で10MHz(8.0MHz)に近い値を選ぶと、153.6kHz×64＝9.8304MHz（153.6kHz×52＝7.9872）という数値が算出できる。

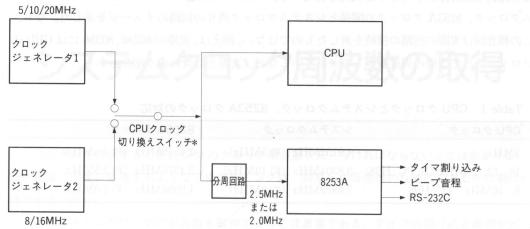
システムクロック周波数は、**システム共通域0000:0501Hのbit7**に示されている(Table 2参照)。この共通域を参照して値を返すプログラムをList 1に示した。

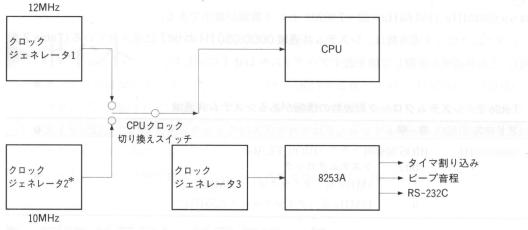
Table 2　システムクロック周波数の情報があるシステム共通域

アドレス	意　味	
0000:0501H	BIOS制御用フラグ（BIOS_FLAG）	
	bit7	システムクロック
	1	8MHz系（タイマクロック2MHz）
	0	10MHz系（タイマクロック2.5MHz）

処理の手順

1. システム共通域からシステムクロック周波数を取得する

PC-9801RX/DX/NS、PC-H98を除く

*クロックジェネレータが1個で切り換えスイッチがないものもある

PC-9801RX/DX/NS

*NSにはクロックジェネレータ2はない

PC-H98

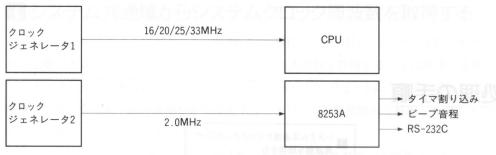



Figure 1　クロック供給の概念図

List 1　システムクロックを取得する

```
;  システムクロックを取得するプログラム例
        push    es
        mov     ax,0000H
        mov     es,ax
        test    es:[0501H],BYTE PTR 80H
        pop     es
        jz      (2.5MHz)
        jmp     (2MHz)
```

▶ワンポイント

■I/Oポートを利用したシステムクロックの取得

システムクロック周波数はI/Oポートから取得することもできる。I/Oポート42Hのbit 5が0ならば5MHz系、1ならば8MHz系である。

しかし、この方法にはいくつかの欠点がある。まず、PC-9801初代ではこのビットは未定義である。また、ハイレゾモードとPC-H98のノーマルモードのフルセントロニクスモードでは、プリンタステータスの入力がこのポートに割り当てられている。そのため、これらの機種にも対応させる場合には、同時に機種判定を含めた繁雑な処理が必要になる。

結局、普通の用途ではI/Oポートを参照するよりはシステム共通域を調べるほうが簡単といえるだろう。

ライブラリ

GetSysClk

解説　システムクロックを取得する。書式はつぎのとおり。

int **GetSysClk**(void)

戻り値　システムクロックの系列を返す。値と意味はつぎのとおり。

値	システムクロックの系列
0	5MHz系
1	8MHz系

サンプル　CPUTEST.C

ソフトウェアによるリセット

リセットボタンを押さずに、プログラムからリセット動作をさせたいことがある。ここでは、ソフトウェアによってリセットとほぼ同じ動作をさせる方法を解説する。

ここで解説するリセットは、CPUがリセット時に実行するルーチンにジャンプするだけで、厳密にはリセットボタンによるリセットとは動作が異なる。しかし、ほとんどの場合、リセットボタンによるリセットとほぼ同等の効果を得ることができるが、周辺機器のリセットなどは行われないので、リセットボタンによるリセットと異なる動作をすることがある。

▶ポイント

- PC-9800シリーズでは、厳密に考えるとソフトウェアからリセットボタンを押すのと完全に同等な動作をさせることはできない。ソフトウェアによってシステム全体のリセット信号線をアクティブにするようなハードウェアが備わっていないためである。
- CPUにリセット時の処理ルーチンを実行させることで、リセットボタンを押したときとほぼ同等な動作をさせることはできる。
- 80286以降のCPUを搭載した機種には、I/Oポート経由でCPUのリセット端子をアクティブにする回路（RESETポート）が組み込まれている。仮想86モードでは、このI/Oポートに出力を行ってリセットしないとリセットできない場合がある。
- 例外的に、PC-22359801VM21だけはV30のみを搭載しているにもかかわらず、RESETポートを備えている。ところが、他機種と同じ様にしてこのポートでリセットしようとするとハングアップしてしまう。この現象を回避するため、V30動作時にはどの機種でもRESETポートによるリセットを行わないようにする。
- RESETポートを搭載した機種では、リセットボタンが押されたのか、ソフトウェアが意図的にI/Oポート経由でリセットしたのかを区別するためのフラグがシステムポートに用意されている。これをリセットボタンが押されたときと同じ状態に設定しておくことにより、リセット時の処理ルーチンを実行させる。

▶プログラミングテクニック

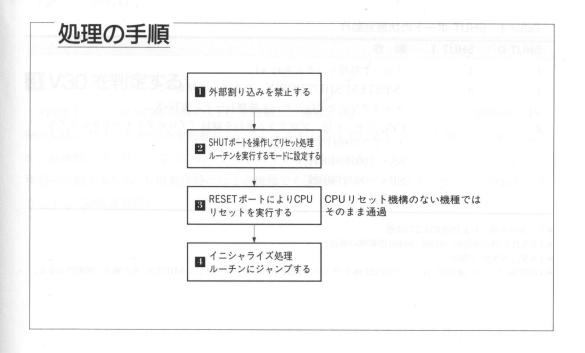
1 外部割り込みを禁止する

タイマなどが動作していると、リセットルーチンの実行中に割り込みがかかって誤動作する可能性があるので、それを防ぐために最初に割り込みを禁止する。プログラムはList 1のとおりである。

List 1　外部割り込みの禁止

```
cli                                          ; 割り込み禁止
```

2 SHUTポートを操作して、リセット処理ルーチンを実行するモードに設定する

80286以降のCPUを搭載した機種には、CPUをリセットする機能がI/Oポートに備わっている(CPUをリセットする機能であって、システム全体をリセットする機能ではない)。また、8086、V30、V33、V50だけを搭載した機種にはそのような機能は存在しない。そこで、ここではとりあえずリセット機能を持つ機種のために、リセット機能のための処理を行う。リセット機能を持たない機種ではこのI/Oポートを操作しても何も起らないのでそのまま実行してしまい、そのあとにリセット時の処理ルーチンにジャンプする。

さて、80286は、一度プロテクトモードに移行してしまうとソフトウェアではリアルモードに戻ることができない。パーソナルコンピュータで利用するときこれでは困るので、CPUリセットでリアルモードに戻ることを利用して、PC-9801には外部ハードウェアでCPUのリセット端子をアクティブにする回路が組み込まれている。

この回路が組み込まれた機種でI/OポートF0Hへの出力を行うと、CPUのリセット端子がアクティブになる。しかし、CPUはいったんリセットされてしまうと果たしてリセットボタンが押されてリセット信号が発生したのか、プログラムが意図的にI/Oポートを操作してCPUリセットを行ったのか自分自身では区別がつかなくなってしまう。そこで、システムポートのポートCのbit 7とbit 5をその区別をするためのフラグ(SHUT 0、SHUT 1)として用いている。

BIOSのリセット処理ルーチンの最初の方に、このフラグをチェックする部分がある。リセットボタンが押されてシステム全体がリセットされた状態では、SHUT 0、SHUT 1とも1が読み出される。これはつぎのような特性を利用している。SHUTポートの存在するシステムポートの8255Aは、リセットされるとすべてのポートが入力モードになる。そこで、ポートCのD7端子とD5端子(SHUT 0とSHUT 1)をプルアップしておけば、リセットスイッチによるリセットのときには必ず両方とも1が読み出せるようになる。ROM内のイニシャライズルーチンがポートCを出力モードに設定してからは、SHUT 0、SHUT 1は任意の状態を保持できる2ビットのメモリとして使うことができる。イニシャライズ終了後はやはりSHUT 0、SHUT 1とも1が設定されているので、OUT F0Hを行うとリセット動作が行われる。SHUT 0に0を設定した状態でOUT F0Hを行うとプログラムの実行が継続して行えるようになっている。SHUT 0、SHUT 1の設定値の意味をTable 1に示す。また、SHUT 0、SHUT 1ポートアドレスをTable 2に示す。

Table 1　SHUTポートの状態別動作

SHUT 0	SHUT 1	動　作
1	1	リセット処理ルーチン実行 *1
1	0	「SYSTEM SHUTDOWN」を表示して停止 *2 メモリクリアせずにリセット処理ルーチン実行*3
0	×	CPUリセット後、プログラム実行を継続(プロテクトモードからリアルモードへの移行などに利用可) *4 SS ← [0000:0406H] SP ← [0000:0404H] RETF

*1 リセット後、および通常はこの状態
*2 UX21以降の80286、80386、80486搭載機の場合
*3 VX2、VX21の場合
*4 仮想86ドライバ使用時、ほとんどの仮想86ドライバはF0Hポートをトラップして SHUT 0=0の場合に無動作となる。

Table 2　SHUT ポートのアドレス

I/O ポート	意　味	
35H	システムポート 8255A ポート C SHUT0、1 ポートの読み出し	
	ビット	ポート
	bit 7	SHUT 0
	bit 5	SHUT 1
37H	8255A コントロールレジスタ システムポート 8255A ポート C SHUT0、1 ポートの設定	
	値	意　味
	0EH	SHUT 0 ← 0
	0FH	SHUT 0 ← 1
	0AH	SHUT 1 ← 0
	0BH	SHUT 1 ← 1

ここでは、リセット処理ルーチンを実行させるために SHUT 0=1、SHUT 1=1 と設定する。プログラムは List 2 のようになる。

List 2　SHUT ポートをイニシャライズを指定するように設定

```
mov     al,0FH          ;SHUT 0 ← 1
out     37H,al
mov     al,0BH          ;SHUT 1 ← 1
out     37H,al
```

3 V30 を判定する

RESET ポートは 80286 以降の CPU を搭載した機種に備わっているものだが、例外的に PC-9801VM21 だけは V30 のみを搭載しているにもかかわらず RESET ポートを備えている。ところが、他機種と同じ様にしてこのポートでリセットしようとするとハングアップしてしまう。この現象を回避するため、V30 動作時にはどの機種でも RESET ポートによるリセットを行わないようにする（List 3 参照）。

List 3　V30 の判定

```
; V30 のときは OUT F0H をスキップする
                mov     ax,0000H
                mov     ds,ax
                test    ds:[0501H],BYTE PTR 40H
                jnz     [5]
```

4 RESET ポートにより CPU リセットを実行する

SHUT 0、SHUT 1 とも 1 が設定された状態で OUT F0H を行うと、80286 以降の CPU を搭載した機種ではリセット動作が行われる（Table 3 参照）。

Table 3　CPU　RESET ポート

I/O ポート	意　味
F0H	CPU RESET ポート 値　　意　味 00H　80286/386 を選択してリセット 07H　V30 を選択してリセット

プログラムは List 4 のようになる。

List 4　CPU リセット実行

```
;       80286、80386、80486 搭載機種では CPU RESET ポートに出力すると、CPU の
;       RESET 端子のみアクティブになる。
                mov     al,0
                mov     dx,00F0H
                out     dx,al           ;CPU RESET
                mov     cx,1000H
                loop    $               ;リセットがかかるまで待つ
```

5 イニシャライズ処理ルーチンにジャンプする

80286 以降の CPU を搭載していない機種には CPU リセットポートが備わっていないので、OUT F0H は素通りする。そこで、そのあとにリセット時の処理ルーチン（FFFF0H 番地）にジャンプする命令を置いて、これらの機種でもリセット動作が行えるようにする。

CPU のリセット端子がアクティブになると、8086・V30・V50・V33 は FFFF0H 番地から実行を始める。ここは BIOS ROM 領域で、リセット処理のプログラムはこの番地から始まっている。

つまり、JMP FFFF:0000H を実行すれば、リセットと同じ動作をさせることができるのである(ワンポイントの「リセット時の実行開始番地」参照)。

ただし、MASM では JMP FFFF:0000H のような記述はできないので、List 5 のように表現する必要がある。

List 5　F0H ポートを持たない機種で、リセット処理ルーチンにジャンプするプログラム

```
                jmp      FAR PTR ResetEntry        ;JMP FFFF:0000H

ResetSegment    SEGMENT  AT 0FFFFH              ; セグメント FFFFH を定義
                ASSUME   cs:ResetSeg
                ORG      0000H
ResetEntry      PROC     FAR                    ;0000H 番地のラベル
ResetEntry      ENDP
ResetSegment    ENDS
```

RESET ポートによってリセットした場合でも、実際に行われるのは FFFF:0000H へのジャンプである。ただ、仮想 86 モードなどで、RESET ポートによってリセットしないとリセット動作できない場合がありうるため、より確実にするのである。

▶ワンポイント

■リセット時の実行開始アドレス

リセット時の実行開始アドレスは実際には CPU により異なっている。リセット時の実行開始アドレスをより一般的に表現すると、「CPU の物理アドレス空間－16 バイトの位置」となる。つまり、8086 の物理アドレス空間は 1M バイトなので FFFF0H 番地、80286 と 80386SX、SL (98) は 16M バイトなので FFFFF0H、80386DX や 80486 は 4G バイトなので FFFFFFF0H ということになる。これは、ROM を全メモリ空間の最後部に置くシステムを設計できるようにするためである。

80286、80386、80486 のリセット時の実行開始アドレスは 1M バイトを超えた位置だが、実行モードはプロテクトモードではなく、特殊なリアルモードである。一度 FAR ジャンプすると 1M バイト以内のアドレスでのみ実行できる通常のリアルモードに移行する。リセット時、80286、80386、80486 では CS＝F000H、IP＝FFF0H で実行が始まるので、必要ならば NEAR ジャンプできる範囲内 (64K バイト以内) でリアルモードに移行するための準備をすることができる。なお、8086、V30、V50、V33 では CS＝FFFFH、IP＝0000H でプログラムの実行が始まる。

PC-9800シリーズでは、Figure 1に示したような箇所にBIOS ROMが出現するようになっている。80286、80386、80486を搭載した機種でもすぐにFARジャンプを行ってリアルモードに移行している。このため、8086搭載機と同じようにJMP FFFF:0000Hでリセット動作を行わせることができる。

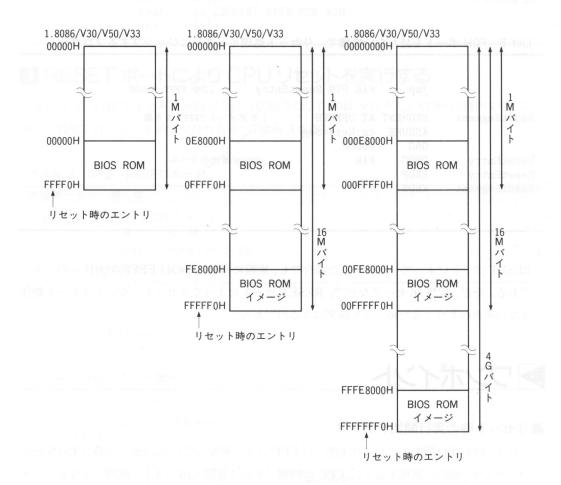



Figure 1 BIOS ROMの出現する位置

■仮想86ドライバ組み込み時の動作

80386や80486の仮想86モードを利用したドライバが組み込まれている場合、ドライバによってはJMP FFFF:0000Hでリセットできないことがある。理由は、おそらくイニシャライズルーチンの中のプロテクトモード命令を仮想86ドライバがトラップするためだと思われる。逆に、JMP FFFF:0000Hで正常にリセットされる仮想86ドライバは、そのためのなんらかの処理を行っていると考えられる。

仮想86ドライバは80286・80386・80486搭載機種に備わっているF0Hポートの監視も行っている。SHUT 0=0でソフトウェアでリセットした後にプログラム実行を継続するような動作を禁

止するためである。OUT F0H でソフトウェアでリセットされると当然 CPU は仮想 86 モードから抜け出してしまうので、そのままでは正常な動作が継続できないからだ。ところが、筆者が確認した範囲のドライバでは、SHUT 0=1 のときには実際に OUT 命令が実行された。つまり、実際にシステムをリセットする動作は許容しているわけである。

というわけで、仮想 86 モードで動作中でも SHUT 0 や SHUT 1 でリセット動作を指定している場合は OUT F0H で正常にシステムのリセットが行えることになる。

■ソフトウェアによるリセットと真のリセットの相違点

リセットルーチンを実行させて行う擬似リセット動作では、実際に本体のリセットボタンを押したときのリセット動作とは違って、バスのリセット信号線がアクティブにならない。そのため、本体内蔵の周辺機器や拡張スロットのボードがハードウェア的に初期化されない。本体内蔵のペリフェラルデバイス類のほとんどは BIOS ROM で初期化が行われるのだが、拡張スロットに実装した増設ボードは本体の BIOS ROM では初期化が行われない（例外として、サウンドボードを内蔵していない機種でその初期化を行うものもある)。ハードウェア的な初期化を行わないためになんらかの障害が起ることはあまりないが、使用するボードやリセットのタイミングによっては問題が起る可能性もあるということは知っておいたほうがよいだろう。

ソフトウェアによるリセットではバスのリセット信号線がアクティブにならないという点を逆に利用することもできる。例えば、EMS メモリ上に BIOS の内容を書き込んでソフトウェアリセットし、EMS メモリ上で BIOS を動作させるようなことが可能になる。サウンド ROM を切り離すことができない PC-9801UV11 で EMS メモリと SASI ハードディスクを同時に使用できるようにする、EMSHD というフリーソフトウェアは、このアイデアを利用したものである。

ライブラリ

CpuReset

解説　PC-9801 をリセットする。書式はつぎのとおり。

```
void CpuReset(void)
```

戻り値　なし

サンプル　RESET.C

テキストVRAM編

テキストVRAMの基礎知識

PC-9800シリーズは、初代から文字表示用のテキストVRAMをもっている。これを使用して画面に文字を表示する方法は、グラフィックVRAMを使用した文字表示より簡単で高速である。初代のPC-9801の頃は他のコンピュータに比較して圧倒的に文字表示が速く、ワープロなどのアプリケーションが普及した原因の1つでもあった。

このテキストVRAMは通常のメモリと同じようにCPUからバイト単位、あるいはワード単位で読み書きできるようになっている。ただし、テキストVRAMはメインメモリに比べてアクセスに必要な時間はかなり長い。

また、テキストVRAMはテキスト表示用とアトリビュート用にわかれている。テキスト表示用の部分には表示する文字コードを設定する。アトリビュート用の部分には表示する文字の色や反転状態、点滅などの属性を文字ごとに設定する。

実際のテキスト画面の制御にはGDC（グラフィックディスプレイコントローラ）が使用されている。GDCとしては、NECのμPD7220が使われている。カーソル表示などはテキスト用のGDCの機能を利用しているので、カーソル形状の変更など、CRT BIOS内に処理が用意されていないことを行いたい場合はテキスト用GDCを直接操作する必要がある。

ここでは、テキストVRAMを利用する方法や注意点、テキストVRAM関係の制御方法などについて解説する。

テキストVRAMの構造

テキストVRAMはTable 1のように、ノーマルモードでは物理アドレスのA0000HからA3FFFHまでのアドレス空間に、ハイレゾモードではE0000HからE3FFFHまでのアドレス空間に割り当てられている。

ノーマルモードとハイレゾモードではテキストVRAMのメモリアドレスが異なったり、1画面に表示できる行数が異なったりしているが、その他の点については大きな違いはない。

テキストVRAMのアトリビュート部の最後部には不揮発性メモリとして利用されている部分が8バイトある。この部分は画面表示の範囲外になっており、通常はハードウェア的に書き込みが禁止されている。したがって、通常のテキストVRAMの操作では、とくに不揮発性メモリに対して考慮する必要はない。

Table 1　テキスト VRAM のアドレス

動作モード	テキスト表示用のアドレス	アトリビュート用のアドレス	画面の構成 (桁数×行数)
ノーマル	A0000H～A1FFFH	A2000H～A3FFFH	80 × 20 or 80 × 25
ハイレゾ	E0000H～E1FFFH	E2000H～E3FFFH	80 × 25 or 80 × 31

ノーマルモードとハイレゾモードではテキスト VRAM のメモリアドレスが異なっているため、両方のモードに対応するプログラムを作成する場合は、現在の動作モードからテキスト VRAM のメモリアドレスを決定する必要がある。現在の動作モードはシステム共通域にある **BIOS 制御フラグ（0000：0501H）**で判定できる（Table 2 参照）。

Table 2　BIOS 制御フラグ

アドレス	意　味	
0000:0501H	BIOS 制御フラグ 1（BIOS_FLAG）	
	bit3	ハイレゾモードとノーマルモードの判別
	1	ハイレゾモード
	0	ノーマルモード

ノーマルモードとハイレゾモードではディスプレイの解像度は異なるが、テキスト VRAM の操作に関してはメモリアドレスが異なることと、1 画面に表示できる行数が異なることを除けば大きな相違はない。したがって、この 2 点さえ注意すれば、テキスト VRAM への表示処理に関してはノーマルモードとハイレゾモードの両方に対応したプログラムにすることは容易である。

テキスト VRAM はサイズ的には 2 画面分が用意されているが、MS-DOS 上では日本語 FEP などが 2 画面目を使用することが多いので、実際にアプリケーションプログラムで使用できるのは 1 画面目のみと考えたほうがよい。また、テキスト VRAM は GDC による画面分割という機能も持っているが、これも日本語 FEP などが使用するので、アプリケーションプログラムでは画面分割は利用できないと考えたほうがよい。

テキスト画面の状態

PC-9800 シリーズではノーマルモードの場合、1 画面の表示行数として 20 行と 25 行が設定できる。ハイレゾモードの場合、25 行と 31 行が設定できる。また、MS-DOS ではファンクションキー表示がされているかどうかによって、プログラムで利用できる行数が変わる。そのため、テキスト VRAM を直接操作するプログラムを作成する場合は、表示行数やファンクションキーの表示状態をプログラム起動時に設定するか、現在の表示行数やファンクションキーの表示状態を調べて、それに合わせて動作させることが必要になる。

現在の画面の表示行数を調べるには、エスケープシーケンスを利用する方法、CRT BIOSを使用する方法、MS-DOSのワークエリアを調べる方法などがある。CRT BIOSでは画面の表示行数は取得できるが、ファンクションキーの表示状態を調べることはできない。また、古いMS-DOSではCRT BIOSで返される画面表示行数と実際のMS-DOSでの表示行数が違うことがある。両方の値を一度に調べる方法として、MS-DOSのワークエリアを調べる方法がある(Table 3参照)。

Table 3 MS-DOSのワークエリア

アドレス	意 味
0060:0112H	スクロール範囲下限 画面の上端を0とした行位置で示される。スクロール範囲下限の直下の行にファンクションが表示される。通常はファンクションキー表示部を除いた画面表示行数−1と等しい

テキストVRAM関連I/Oポート

テキストVRAMに関係するI/OポートとしてはTable 4のように、CRTコントローラ、テキストVRAM用のGDC、モードレジスタ、ユーザ定義文字用のI/Oポートなどがある。

CRTコントローラは主にテキスト画面のライン数やスムーススクロールの制御などに使用されている。テキストVRAMのハードウェア的な制御は、大部分をテキストVRAM用のGDCで行っている。

PC-98LT/HAにはテキストVRAMが存在しないため、これらのテキストVRAMに関係したI/Oポートは使用できない。

モードレジスタ(I/Oポート68H)は書き込む値によって機能が異なる。モードレジスタへの設定値と機能に関して、テキストVRAMに関係のあるものはTable 5のような対応になっている。モードレジスタは、Table 5以外にもグラフィック画面の制御に関係するものやメモリスイッチに関するものなどの設定があるが、ここでは省略している。

Table 4　テキストVRAM関連のI/Oアドレス

I/Oポート	Read/Write	内　容
GDC関係のポート		
60H	Read	GDCのステータス
60H	Write	GDCのパラメータ
62H	Read	GDCのデータ
62H	Write	GDCのコマンド
68H	Write	モードレジスタ
6CH	Write	ボーダーカラー
CRTコントローラ関係のポート		
70H	Write	キャラクタ位置ライン数
72H	Write	ボディーフェースライン数
74H	Write	キャラクタライン数
76H	Write	スムーススクロールライン数
78H	Write	スクロールエリア上辺位置行数
7AH	Write	スクロールエリア行数
ユーザ定義文字関係のポート		
A1H	Write	文字コード第2バイト
A3H	Write	文字コード第1バイト
A5H	Write	ラインカウンタ
A9H	Read	CGリードパターン
A9H	Write	CGライトパターン

Table 5　モードレジスタへの設定値

設定値	機能（*はデフォルト）
00H	アトリビュートのD4はバーティカルライン*
01H	アトリビュートのD4は簡易グラフ
04H	80文字/行モード*
05H	40文字/行モード
06H	ANKフォントは6×8ドット
07H	ANKフォントは7×13ドット*
0AH	KCGコードアクセスモード*
0BH	KCGドットアクセスモード
0EH	画面表示禁止
0FH	画面表示許可*

VSYNCについて

テキスト画面の表示はCPUの動作とは無関係にハードウェアで行われている。そのため、画面の書き換えや表示開始、スクロールなどを画面の表示と無関係のタイミングで行うと、一瞬だけ表示が乱れたり、スクロールが滑らかに見えなかったりすることがある。

ディスプレイ出力には表示期間とブランク期間があり、これが1秒間に約60回ほどの頻度で繰り返されている。ブランク期間中にテキスト画面の操作を行えば、表示の乱れを防ぐことができる。また、画面表示のタイミングに合わせてスクロール処理を行うと、1行単位のスクロールであっても、とても滑らかにスクロールしているように見える。

ブランク期間をソフトウェアで直接知ることはできないが、Figure 1のように、ブランク期間には必ずVSYNC（垂直同期）が1になる期間があり、ブランク期間以外でVSYNCが1になることはない。しかも、VSYNCはテキストVRAM用のGDCのステータス（I/Oポート60H）のbit 5で状態を調べることができる。

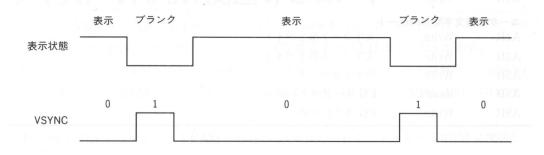



Figure 1　表示状態とVSYNCの関係

したがって、VSYNCが1になるのを待った後で、画面表示の操作を行えば、画面表示の乱れを防ぐことができる。

VSYNCが1になるのを待つプログラムはList 1のようになる。

List 1　VSYNCを待つルーチン(1)

```
vsync:
        in      al,60h          ; GDCのステータスを読み出す
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        test    al,20h          ; VSYNC中か
        jz      vsync
```

ただし、このプログラムはVSYNC検出直後にハードウェア割り込みなどが発生した場合などの事は考慮していないため、つぎに実行する処理をVSYNC期間中あるいはブランク期間中にするためには、List 2のvsync_waitルーチンのように少し工夫が必要となる。

List 2 VSYNCを待つルーチン(2)

```
vsync_wait:
        jmp     short vsync
vwait:
        popf
        jmp     $+2                     ; リカバリタイム兼他の割り込み処理用
        jmp     $+2
vsync:
        pushf
        cli                             ; 割り込みを禁止
        in      al,60h                  ; GDCのステータスを読み出す
        test    al,20h                  ; VSYNC中か
        jz      vwait
          :                             ; ブランク期間中に行いたい処理
        popf
```

このプログラムの場合、ブランク期間中に行いたい処理は、割り込みが禁止された状態で実行されるため、実行時間の短いものに限られる。

VSYNC期間中あるいはブランク期間中であることを保証するためには、このような割り込みフラグの制御や、場合によってはVSYNCの再チェックが必要になる。

VSYNC期間を待つ処理での注意点は、割り込み禁止状態のままVSYNC期間になるのを待ってはいけないことである。割り込み禁止状態のままVSYNC期間を待つと、RS-232Cの割り込みやタイマ割り込みなどのハードウェア割り込みが無視されてしまうからである。RS-232Cの割り込みが無視されると受信データの欠落が発生することになり、タイマ割り込みが無視されると、タイマ割り込みを使用しているソフトウェアの時間計測が正しくできないことになる。

PC-98LTやPC-98HAではGDCが搭載されていないので、VSYNCを待つプログラムは当然動かない。これらの機種でも擬似的なVSYNC割り込みが定期的に発生するようになっているが、本来のVSYNCとはまったく無関係である。

漢字コードの変換

テキスト VRAM に全角文字を書き込む場合は JIS 漢字コードを基にした文字コードを使用する。しかし、MS-DOS では漢字コードにシフト JIS 漢字コード（MS 漢字コード）を使用しているため、プログラム中ではシフト JIS 漢字コードを扱い、テキスト VRAM に書き込む時にシフト JIS 漢字コードから JIS 漢字コードへ変換することになる。

VRAM などを利用する際には、頻繁に利用することになるので、List 3 と List 4 のような変換プログラムを用意する。ここでは、そのアルゴリズムなどには詳しく触れないが、自分で作成される場合には、『MS-DOS プログラマーズハンドブック』（アスキー出版局刊）などを参考にするとよいだろう。漢字コードの変換処理は、小規模で一度正しく動作するものを作成しておけば後で変更する必要はまずないので、可読性の良さより、コンパクトさやスピードをとった方がよい。なお、実際には、コードの変換以外に正しいコードであるかどうかの確認も必要となる。

List 3　JIS 漢字コードをシフト JIS 漢字コードにするプログラム

```
; JIS 漢字コードからシフト JIS 漢字コードへの変換
; AH: JIS 漢字コードの 1 バイト目
; AL: JIS 漢字コードの 2 バイト目
        add     ah,21h          ; 21H～5EH を 42H～7FH に変換
                                ; 5FH～7EH を 80H～9FH に変換
        sar     ah,1            ; 42H～7FH を 21H～3FH に変換
                                ; 80H～9FH を C0H～CFH に変換
        jnc     conv            ; シフト前の AH が偶数なら conv へ
        add     al,5eh          ; 21H～7EH を 7FH～DCH に変換
conv:
        add     al,0a0h         ; 21H～5FH を C1H～FFH に変換しキャリーを 0 にする
                                ; 60H～DCH を 00H～7CH に変換しキャリーを 1 にする
        adc     al,7fh          ; C1H～FFH を 40H～7EH に変換
                                ; 00H～7CH を 80H～FCH に変換
        xor     ah,20h          ; 21H～3FH を 01H～1FH に変換
                                ; C0H～CFH を E0H～EFH に変換
        or      ah,80h          ; 01H～1FH を 81H～9FH に変換
; AH: シフト JIS 漢字コードの 1 バイト目
; AL: シフト JIS 漢字コードの 2 バイト目
```

List 4　シフトJIS漢字コードをJIS漢字コードにするプログラム

```
; シフト JIS 漢字コードから JIS 漢字コードへの変換
;AH: シフト JIS 漢字コードの1バイト目
;AL: シフト JIS 漢字コードの2バイト目
        add     ah,ah           ;81H～9FH を 02H～3EH の偶数に変換
                                ;E0H～EFH を C0H～DEH の偶数に変換
        sub     al,1fh          ;40H～7EH を 21H～5FH に変換
                                ;80H～9EH を 61H～7FH に変換
                                ;9FH～FCH を 80H～DDH に変換
        js      conv            ;80H～DDH の場合は CONV へ
        cmp     al,61h          ;21H～5FH の場合はキャリーフラグを1にセット
                                ;61H～7FH の場合はキャリーフラグを0にセット
        adc     al,0deh         ;AL が 21H～5FH なら 00H～3EH に変換
                                ;AL が 61H～7FH なら 3FH～5DH に変換
conv:
        add     ax,1fa1h        ;AL が 00H～5DH なら A1H～FEH に変換し AH に 1FH を加算
                                ;AL が 80H～DDH なら 21H～7EH に変換し AH に 20H を加算
        and     ax,7f7fh        ;JIS 漢字コードにするためのマスク処理
;AH: JIS 漢字コードの1バイト目
;AL: JIS 漢字コードの2バイト目
```

テキストVRAMからの文字の読み出し

テキスト画面に表示されている文字は、テキストVRAMを直接参照すれば知ることができる。ただし、テキストVRAMに格納されている文字は、半角や全角の右半分や左半分であることを示すために若干の変換がほどこされている。

ここでは、VRAMから文字コードを取得する方法を解説する。また、本節では、VRAMの構造やデータの格納形式もあわせて解説している。アトリビュートなどを操作したい人にとっても参考になるだろう。

▶ポイント

- PC-9800シリーズのVRAMの構成は、全角文字とANK文字（1バイト系半角文字）を混在させるために、ANK文字1つにつき2バイトが割り当てられている。
- ANK文字は下位バイトにASCIIコードが納められている。
- 全角文字はJIS漢字コードに多少の変換を加えた文字コードが納められている。
- 全角文字はANK文字2つ分を占有するため、下位バイトのbit7が0か1かで、左半分と右半分を識別する。

▶プログラミングテクニック

1 VRAM上のアドレスを計算する

テキストVRAMはテキスト部とアトリビュート部にわかれており、Figure 1のように、それぞれ8Kバイトを占有している。

テキスト部もアトリビュート部も約2画面分の大きさがあるが、MS-DOSでは2画面目の一部をFEPが使用することが多いので、アプリケーションプログラムでは通常は1画面分しか利用できない。

アトリビュート部は、奇数アドレスは使用されていない。ただし、PC-H98シリーズの拡張アトリビュートモードでは奇数アドレスも使用されている。

テキスト部は半角1文字につき2バイトを占有する。テキストVRAMに設定する値は表示す

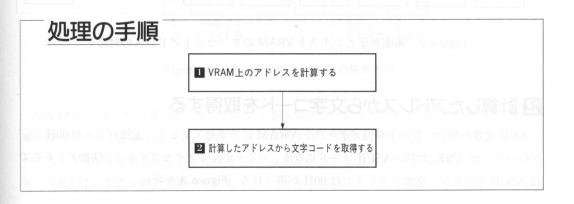
Figure 1　テキストVRAMのメモリマップ（括弧内はハイレゾモードのアドレス）

る文字がANK文字か、ひらがなや漢字などの全角文字か、2バイト系半角文字かによって異なる。文字の反転や色を示すアトリビュート部もテキスト部と同じように半角1文字につき2バイトが割り当てられている。

画面上の1行は半角で80文字分であり、テキストVRAMのテキスト部は1行につき160バイトが割り当てられる。実際の画面上の半角1文字単位での表示位置とテキストVRAMの先頭からのオフセットアドレスとの関係はFigure 2のようになっている。

画面上の桁位置X、行位置YとテキストVRAMのテキスト表示部分のオフセットアドレスADRとの関係は次のような式になる。

$$ADR = (Y \times 80 + X) \times 2$$

つまり、1文字右のアドレスは2増え、1行下のアドレスは160増える。上の式をプログラムに

すると、List 1 のようになる。

文字に対応するアトリビュート部のアドレスは、ここで計算したアドレスに 2000H を加えた値になる。アトリビュートをアクセスする場合は、テキスト部のオフセットアドレスに 2000H を加えるか、セグメントのアドレスをアトリビュート部のセグメントに一致させてオフセットをテキスト部と同じままにするかのどちらかである。

List 1　テキスト VRAM のアドレス計算

```
; テキスト VRAM のアドレス計算
; AX: 行位置 Y
; BX: 桁位置 X
        mov     cl,80
        mul     cl
        add     bx,ax
        add     bx,bx
; BX: オフセットアドレス ADR
```

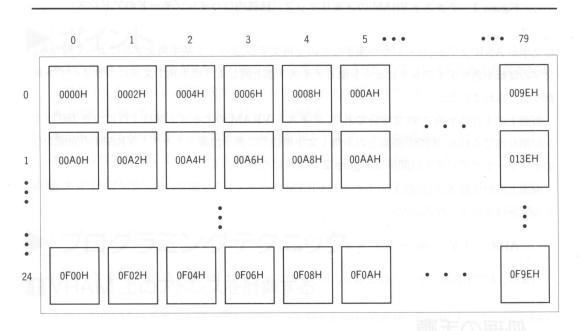



Figure 2　画面表示とテキスト VRAM のオフセットアドレスの対応

2 計算したアドレスから文字コードを取得する

ANK 文字の場合、ワード単位でテキスト VRAM にアクセスすると、上位バイトが 00H で、下位バイトが ANK 文字の ASCII コードになる。バイト単位でアクセスすると、偶数アドレスでは ASCII コードが、奇数アドレスでは 00H が得られる（Figure 3 参照）。

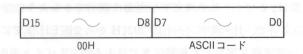



Figure 3　VRAM での ANK 文字の格納形式

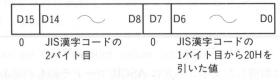
PC-9801VM 以前の機種では、ASCII コードの 60H のアクサングラーブ（ ‘ ）は表示されない。また PC-9801VX などでは字形がアクサングラーブではなくバッククォートになっている。ASCII コードの FCH のバックスラッシュ（\）も PC-9801VM 以前の機種では表示されない。画面に表示できる ANK 文字の詳細については『テクニカルデータブック』を参照して欲しい。

ひらがなや漢字、ユーザ定義文字などの全角文字は、JIS 漢字コードを基準とした文字コードがテキスト VRAM に書き込まれている。

ワード単位でアクセスする場合は、下位バイトには JIS 漢字コードの 1 バイト目から 20H を引いた値が、上位バイトには JIS 漢字コードの 2 バイト目が格納されている。

JIS 漢字コードをワード単位で扱う場合と、テキスト VRAM をワード単位でアクセスした場合とでは、上位バイトと下位バイトが入れ替わっているので注意が必要である。

全角で表示する文字は左半分と右半分でそれぞれ 1 ワードを使用している。全角文字の左右を判別できるようにするために、通常は左半分の bit7 は 0 に、右半分の bit7 は 1 になっている。また、bit15 は未使用ビットで 0 になっている（Figure 4 参照）。ただし、bit7 の左右の判別はハードウェア的な仕様ではなくソフトウェア的なものである。

全角文字の左半分

全角文字の右半分

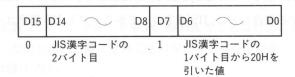



Figure 4　VRAM での全角文字の格納形式

MS-DOS のように内部コードとしてシフト JIS 漢字コードを利用している場合は、シフト JIS 漢字コードから JIS 漢字コードに変換し、さらにテキスト VRAM 用の文字コードに変換してからテキスト VRAM に書き込むことになる（シフト JIS コードと JIS コードの変換については「テキスト VRAM の基礎知識」の節を参照）。

PC-9800シリーズでは、2バイト系半角文字を画面に表示できるようになっている。これはPC-9800シリーズ独自の文字で、JIS漢字コードの2921Hから2B7EHまでに割り当てられている(Figure 5参照)。ユーザ定義文字なども制限付きではあるが、2バイト半角文字と同様に半角幅で表示することができる。

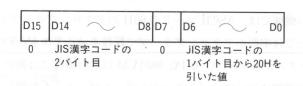



Figure 5　VRAMでの2バイト系半角文字の格納形式

こうした格納形式を考慮に入れて、VRAMのデータから文字コードを取得するプログラムを作成するとList 2のようになる。

List 2　VRAMから文字コードに変換する

```
; es:bx: VRAM上のアドレス
        mov     ax,0a000h       ; テキストVRAMのセグメントアドレス
        mov     es,ax
        mov     ax,es:[bx]      ; テキストVRAMの内容を読み出す
        or      ah,ah           ; ANK文字かチェック
        jz      pend            ; ANK文字なら終了へ
        xchg    ah,al           ; 上位バイトと下位バイトを交換
        add     ah,20h          ; JIS漢字コードに変換
pend:
; AX: 取得した文字コード
```

ANK文字の場合は取得した文字コードはASCIIコードそのものであるから問題ないが、全角文字などの場合はJIS漢字コードで返される。したがって、テキストVRAMの内容をMS-DOSのファイルに変換する場合などは、JIS漢字コードからシフトJIS漢字コードへの変換処理が必要となる。

▶ワンポイント

■PC-98LT/HAのVRAM

PC-98LT/HAはグラフィックVRAMに文字を描画しており、ハードウェア的にはテキストVRAMは存在しないが、メインメモリ上に仮想テキストVRAMが用意されており、CRT BIOSで仮想

テキスト VRAM の先頭アドレスを取得することができるようになっている。

この仮想テキスト VRAM は通常のテキスト VRAM よりサイズが小さく、構造も多少異なるので注意が必要である。テキスト VRAM のテキスト部とアトリビュート部の先頭アドレスの差は、ノーマルモードもハイレゾモードも 2000H だが、PC-98LT や PC-98HA 内蔵の MS-DOS が用意している仮想テキスト VRAM は 10A0H になっている。また、アトリビュート部の構造も画面書き換えの制御のために拡張されている。

テキスト VRAM を直接操作するプログラムを PC-98LT や PC-98HA にも対応させたい場合は、最初に CRT BIOS を使用して仮想テキスト VRAM の先頭アドレスを取得し、画面表示用の CRT BIOS などを使用する必要がある。もちろん、テキスト表示用 VRAM とアトリビュート用 VRAM の大きさ、先頭アドレスの差などが通常の PC-9800 シリーズと異なることに注意しなければならない。

CRT BIOS を使用した文字表示は遅いので、高速性が要求されるプログラムでは直接グラフィック VRAM に文字を描画する方がよいが、文字フォントはメインメモリ中に配置されており、文字コードから文字フォントへのアドレス計算やバンクメモリ切り換えも単純ではない。また、グラフィック VRAM に文字を描画するときでも、仮想 VRAM への書き込みを省略すると日本語 FEP などを使用する場合に問題が起きるので、注意が必要である。

ライブラリ

GetVram

解説　引数で指定された画面上の位置の文字を読み取り、その文字コードを返す。書式はつぎのとおり。

unsigned int **GetVram**(int *colum*, int *line*)

colum　画面上の桁位置（0～79）
line　画面上の行位置（0～画面表示行数−1）

この関数は、ハイレゾモードや PC-98HA などには対応していない。また、指定した桁と行の位置に全角文字の左右のどちらが存在していても、その別は判断しない。

戻り値　引数で指定された画面上の位置の文字コードを返す。文字が半角ならば上位バイトは 0、下位バイトは ASCII コードである。文字が 2 バイト文字ならば上位バイトが JIS コードの 1 バイト目、下位バイトが JIS コードの 2 バイト目になる。

サンプル　VRAM1.C

テキストVRAMへの文字の書き込み

文字表示は、MS-DOSのシステムコールでも行えるが、高速に表示するにはテキストVRAMへ直接文字コードを書き込むという方法をとればよい。ただし、全角文字をテキストVRAMへ書き込む場合は、画面を乱さないために、左右の書き込む順番や消去する順番を考察しなければいけない。ここでは、テキストVRAMに直接文字コードを書き込む方法を、全角文字の扱い方も含めて解説する。

▶ポイント

- ノーマルモードのテキストVRAMのアドレスは、テキスト部はA000：0000H、アトリビュート部はA000：2000Hである。
- ANK文字はASCIIコードを下位バイトに、上位バイトを00Hにする。
- 全角文字は、JIS漢字コードに変換を加え、右半分と左半分の識別用のビットを設定する。
- 書き込み時に画面を乱さないために、全角文字を書き込むときは、左半分、右半分の順序で書き込む方がよい。
- また全角文字を半角文字で消去するときは、右半分、左半分の順序で消去する方が画面の乱れが少ない。

▶プログラミングテクニック

1 VRAM上のアドレスを計算する

表示する座標から、VRAM上のアドレスを計算する。実際の方法は、「テキストVRAMからの文字の読み出し」の1同様である。

2 計算したアドレスに文字を書き込む

PC-9800シリーズのテキストVRAMでは、ハードウェア的な制約により、全角文字の左半分と右半分を自由に独立して表示することはできない。

ただし、ユーザ定義文字やNEC独自の追加文字などは、JIS第1水準文字やJIS第2水準文字

などの全角文字とはハードウェア的に扱いが異なり、制限付きではあるが2バイト半角文字のように半角幅単位でも表示できる。

MS-DOSのコンソール出力などのような汎用的な画面表示では、1文字をテキストVRAMに書き込む際は全角文字の上に重なるかどうかを判定し、重なる場合は元の全角文字を消してから表示する文字を書き込む必要がある。このときに、2バイト系半角文字のチェックも必要になる。全角文字の削除の処理を行うときは、全角文字の右半分か左半分かの判断が必要になる。この処理を行わないと、文字がずれたり書き込んだ文字が表示されなかったりなどの問題が発生する。

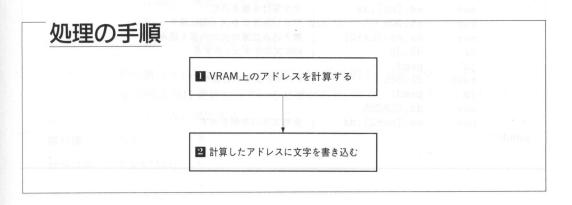
テキストVRAMに全角文字の左右を正しく設定しなかった場合の動作は、機種によって異なる。PC-9801初代/E/F/Mなどの機種では全角文字を左右共に正しく書き込んでいないと文字が半分に崩れて表示される場合がある。一方、PC-9801VM以降の機種ではJIS第1水準やJIS第2水準の文字であれば全角文字の右側にどのような文字コードを書いても文字は半分に崩れたりしない。ただし画面右端にかかる場合は当然ながら半分しか表示されない。しかし、全角文字の左半分だけを消去すると、以降に全角文字が続くと、すべて半角分だけずれて表示されてしまう。

画面消去後に書き込みをする場合など、全角文字の上への重ね書きが起きないことが明らかな場合は、文字表示の際に全角文字の上に重なるかどうかの判定や全角文字の削除の処理をする必要はない。

全角文字を書き込む場合は、全角文字の左半分のワードから先に書き込みをしたほうがよい。全角文字の右半分から書き込みを行うと、左半分が書き込まれるまで画面上の表示がずれることになるからである。

全角文字を消す、つまり2個の半角スペースなどに書き換える場合は、書くときとは逆に全角文字の右半分のワードから先に消したほうがよい。左半分から消すと、右半分が消されるまでの間、画面上の文字が右にずれることになるからである。

画面が乱れないように考慮した、1文字を表示するプログラムをList 1に示す。なお、List 1でアトリビュートへの書き込みをワード単位で行っているのは、PC-H98シリーズの拡張アトリビュートモードへの対応を容易にするためである。

なおこのプログラムでは、あらかじめアトリビュート用にCHRATRという定数を、画面クリ

List 1 VRAMへの文字コードの書き込み

```
; 1文字表示処理
; AX: 表示する文字コード (ANKの場合はAH=00H)
; BX: 表示位置のテキストVRAM上のオフセットアドレス
        push    ax              ; 文字コードを待避
        mov     ax,0a000h       ; テキストVRAMのセグメント
        mov     es,ax
        jcxz    putmain         ; 行頭ならputmainへ
;
        mov     ax,es:[bx]      ; 書き込み位置の内容を読み出す
        or      ah,ah           ; ANK文字かチェック
        jz      putmain         ; ANK文字なら上書きなのでputmainへ
        test    al,80h          ; 全角文字の右側か
        jz      putmain         ; 全角文字の右側でなければputmainへ
        mov     ax,CLRCHR
        mov     es:[bx-2],ax    ; 全角文字の左側を消す
putmain:
        pop     ax              ; 文字コードを取得
        or      ah,ah           ; ANK文字かどうかチェック
        jz      putank          ; ANK文字ならputankへ
        xchg    ah,al           ; 上位バイトと下位バイトを交換
        sub     al,20h          ; オフセット調整
        cmp     al,09h          ; 2バイト半角文字のチェック
        jb      putzen          ; 全角文字ならputzenへ
        cmp     al,0ah          ; 2バイト半角文字のチェック
        jbe     putank          ; 半角文字ならputankへ
putzen:
        mov     es:[bx],ax      ; 全角文字の左側を書く
        or      al,80h          ; 全角右側のフラグを立てる
        mov     es:[bx+2],ax    ; 全角文字の右側を書く
        mov     ax,CHRATR
        add     bh,20h          ; アドレスをアトリビュート部にする
        mov     es:[bx],ax      ; 文字属性を書き込む
        add     bl,02h          ; アドレスを右半分の位置にする
        jmp     SHORT clrcheck
putank:
        mov     es:[bx],ax      ; 文字を書き込む
        mov     ax,CHRATR
        add     bh,20h          ; アドレスをアトリビュート部にする
clrcheck:
        mov     es:[bx],ax      ; 文字属性を書き込む
        sub     bh,20h          ; アドレスをテキスト部に戻す
        mov     dx,es:[bx+2]    ; 書き込み位置の次の内容を読み出す
        or      dh,dh           ; ANK文字かチェックする
        jz      pend
        test    dl,80h          ; 2バイト文字の右側か
        jz      pend
        mov     dx,CLRCHR
        mov     es:[bx+2],dx    ; 全角文字の左側を消す
pend:
```

ア用にCLRCHRという定数を設定してあることを想定している。たとえば、CLRCHRは0020Hにアセンブラの EQU 擬似命令などで設定しておく。また、文字色は白ならCHRATRを04E1Hに設定しておく。汎用のプログラムにするには、EQUではなく引数として受け取れるように改良するとよいだろう。

ライブラリ

PutVram

解説　画面の指定位置に文字を表示する。書式はつぎのとおり。

void **PutVram**（int *colum*, int *line*, unsigned int *ch*）

colum　画面上の桁位置（0〜79）
line　画面上の行位置（0〜現在の画面表示行数−1）
ch　表示する文字コード

表示する文字コードは、文字がANK文字もしくはグラフィック文字の場合は、上位バイトを00Hにして下位バイトにASCIIコードを入れる。全角文字などの2バイト系文字を表示する場合は、上位バイトにJIS漢字コードの1バイト目を、下位バイトにJIS漢字コードの2バイト目を入れる。なお、本プログラムはノーマルモードのみに対応する。

戻り値　なし

サンプル　VRAM1.C

PutStrVram

解説　画面の指定位置に文字列を表示する。書式はつぎのとおり。

void **PutStrVram**(int *colum*, int *line*, char ＊ *str*)

colum　画面上の桁位置
line　画面上の行位置（0〜画面表示行数−1）
str　表示する文字列

桁位置は0から表示する文字列がその行に納まる範囲でなければならない。文字列は00Hで終わっている必要がある。なお、本プログラムはノーマルモードのみに対応する。

戻り値　なし

サンプル　VRAM2.C

カーソル位置の取得

カーソル位置を取得するには、MS-DOSのワークエリアを参照する方法と、GDCの描画アドレスレジスタを読み出す方法がある。

MS-DOSのワークエリアを参照する方法は簡単であるが、テキストVRAMやGDCを直接操作するプログラムを使っている場合など、MS-DOSの画面表示処理を使用していない状態では正しいカーソル位置が得られない可能性がある。

テキスト用のGDCの描画アドレスレジスタを読み出す方法は、テキストVRAMやGDCを直接操作しているプログラムにも有効であり、日本語FEPの多くは、この方法を用いている。

ここでは、GDCの描画アドレスレジスタを参照してカーソル位置を取得する方法を解説する。

▶ポイント

- GDCにCSRRコマンドを発行するとカーソル位置を読み出せる。
- GDCにコマンドを発行するときはGDCのFIFOバッファに空きがあるかどうかを確認する。

▶プログラミングテクニック

1 GDCのFIFOバッファが空になるのを待つ

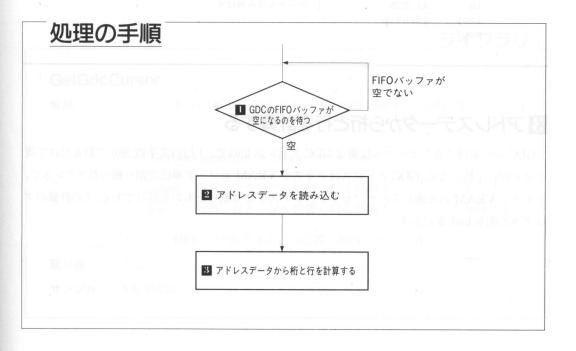
GDCはコマンドの処理に時間がかかるため、CPUからのコマンドを高速に受けられるようにFIFOバッファを持っている。このFIFOバッファがいっぱいの状態では、GDCにコマンドを発行しても無視されてしまう。そこで、FIFOバッファが空になるのを確認してからGDCにコマンドを送るのが、一般的なGDCへのコマンドの発行方法である。

FIFOバッファの状態は、GDCのステータスレジスタ(I/Oポート60H)のbit2で判定できる。bit2が1ならばFIFOバッファは空である。プログラム例をList 1に示す。

List 1　GDC の FIFO バッファが空になるのを待つ

```
        jmp     SHORT gdcloop
gdcwait:
        popf
        jmp     $+2             ; リカバリタイムとウェイト
        jmp     $+2
gdcloop:
        pushf
        cli                     ; 他のプログラムとの競合回避のため割り込み禁止
        in      al,60h          ; GDC のステータスを調べる
        test    al,04h          ; FIFO が空か
        jz      gdcwait         ; 空でなければ gdcwait へ
```

2 アドレスデータを読み込む

GDC からカーソルの位置を読み出すには、最初に CSRR コマンド（E0H）を I/O ポート 62H から GDC に出力する。CSRR コマンドは GDC に対して、これから 5 バイトのデータを読み出すという指示になる。プログラムはコマンドを出力したのと同じ I/O ポート 62H を読んで GDC からのデータを取得する。この最初の 2 バイトがカーソル位置の情報である。これは、最後に GDC に設定されたカーソル位置の GDC アドレスである。GDC アドレスは、2 倍するとテキスト VRAM のオフセットアドレスと一致する。残りの 3 バイトは不要なので、読み捨てる。プログラム例を List 2 に示す。

List 2　GDC に CSRR コマンドを送りカーソルのアドレスを読む

```
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        jmp     $+2
        mov     al,0e0h         ; CSRR コマンド
        out     62h,al          ; GDC にコマンド出力
;
gdcbusy:
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        jmp     $+2
        in      al,60h          ; GDC のステータスを調べる
        test    al,01h          ; コマンド実行終了か
        jz      gdcbusy         ; 終了でなければ gdcbusy へ
;
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        jmp     $+2
        in      al,62h          ; アドレスデータ読み込み
        mov     bl,al           ; アドレスの下位バイトを保存
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        jmp     $+2
        in      al,62h          ; アドレスデータ読み込み
        mov     bh,al           ; アドレスの上位バイトを保存
        mov     cx,3            ; 後続の 3 データを読み飛ばす
gdcskip:
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        jmp     $+2
        in      al,62h          ; データを読み飛ばす
        loop    gdcskip
        popf
```

3 アドレスデータから桁と行を計算する

GDC から取得できたカーソル位置は GDC アドレスなので、1 行の文字数(80)で割るだけで商と余りが行と桁になる。GDC アドレスはテキスト VRAM をワード単位で割り振ったアドレスで、テキスト VRAM の先頭からのオフセットアドレスを 2 で割ったものと同じである。この計算のプログラム例を List 3 に示す。

List 3　読み出したカーソルのアドレスからカーソルの桁と行を計算する

```
        mov     ax,bx           ; 読み出したアドレス
        sub     dx,dx
        mov     bx,80           ; 画面の 1 行の文字数
        div     bx              ; ax に桁を, dx に行を格納
; AX: カーソル位置（桁）
; DX: カーソル位置（行）
```

ライブラリ

GetGdcCursor

解説　テキスト用の GDC から現在のカーソル位置を取得する。書式はつぎのとおり。

void **GetGdcCursor**(int ＊ *colum*, int ＊ *line*)

colum　取得したカーソル位置の桁を格納するアドレスを指定する。
line　取得したカーソル位置の行を格納するアドレスを指定する。

カーソル OFF の状態では正しい位置が取得できない場合がある。

戻り値　なし

サンプル　VRAM2.C

カーソルの大きさの制御

PC-9800シリーズのテキスト画面のカーソルは、テキスト用のGDCがハードウェアによって表示している。GDCを操作すれば、カーソルの表示や消去、カーソルの大きさの変更ができる。

カーソルの表示や消去の機能はCRT BIOSに用意されているが、カーソルの大きさはCRT BIOSからは変更できない。ここでは、GDCに直接CSRFORMコマンドを送ってカーソルの大きさを変更する方法を解説する。

なお、ここで作成するプログラムはノーマルモードを対象とする。また、PC-98LT/HAにはGDCが搭載されていないため、GDCの操作はできない。

▶ポイント

- カーソルの大きさを変えるにはテキスト用のGDCにCSRFORMコマンドを発行する。
- GDCにコマンドを発行するときはGDCのFIFOバッファに空きがあるかどうかを確認する。
- CSRFORMコマンドは表示行数によってパラメータを変える必要がある。

▶プログラミングテクニック

1 表示行数を取得する

カーソルの大きさを変えるにはテキスト用のGDCにCSRFORMコマンドを発行しなければならないが、CSRFORMコマンドでは、カーソルの大きさとともに、1行のライン数も設定しなければならない。ノーマルモードのPC-9800の1画面は400ラインで構成されており、1行のライン数は25行表示では16ラインで、20行表示では20ラインである。つまり、1行のライン数は現在の表示行数に依存する。そこで、現在の表示行数を取得する必要が出てくる。

現在の表示行数を取得するには、システム共通域に用意されている**CRT状態フラグ**0000：053CHを参照する方法がある（Table 1参照）。CRT BIOSを使用して画面表示行数が設定されると、システム共通域に設定値が保存されるようになっている。このCRT状態フラグのbit0が0ならば25行、1ならば20行である。プログラム例をList 1に示す。

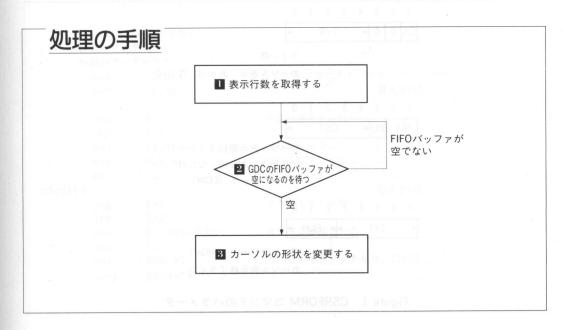
Table 1　システム共通域 CRT 状態フラグ

アドレス	意　味
0000:053CH	テキスト画面ステータスフラグ (CRT_STS_FLAG) bit0　表示行数 1　20行 0　25行

List 1　表示行数の取得

```
sub     ax,ax            ; システム共通域：セグメント 0000H
mov     es,ax
mov     al,es:[053ch]    ; CRT 状態フラグを読み出す
test    al,01h           ; 表示行数は 20 行か
jnz     gdcloop
add     bx,3             ; ポインタを+3 して 25 行用のデータにする
jmp     SHORT gdcloop
```

2 GDC の FIFO バッファが空になるのを待つ

GDC は CPU と比較してコマンドの実行に時間がかかるため、内部に FIFO バッファを持っている。GDC 内の FIFO バッファがいっぱいの場合は、GDC にコマンドを発行しても無視されてしまうので、GDC にコマンドを発行する場合は、FIFO バッファの状態をチェックする必要がある。FIFO バッファの確認には GDC のステータス（I/O ポート 60H）の bit2 を監視すればよい。

プログラム例をList 2に示す。

List 2　FIFOバッファが空になるまで待つ

```
        jmp     SHORT gdcloop
gdcwait:
        popf
        jmp     $+2                 ; リカバリタイム兼ウェイト
        jmp     $+2
gdcloop:
        pushf
        cli                         ; 他のプログラムとの競合回避のため割り込み禁止
        in      al,60h              ; GDCのステータスを調べる
        test    al,04h              ; FIFOバッファが空か
        jz      gdcwait             ; 空でなければgdcwaitへ
```

3 カーソルの形状を変更する

GDCのFIFOバッファが空になったら、次にCSRFORMコマンドをテキスト用のGDCに発行してカーソルの形状を変更する。CSRFORMコマンド自身はGDCのコマンドポート(I/Oポート62H)に出力する。パラメータはI/Oポート60Hに出力する。出力と出力の間にはリカバリタイムをはさむ必要がある。

CSRFORMコマンドには3バイトのパラメータがあり、Figure 1のような構成になっている。

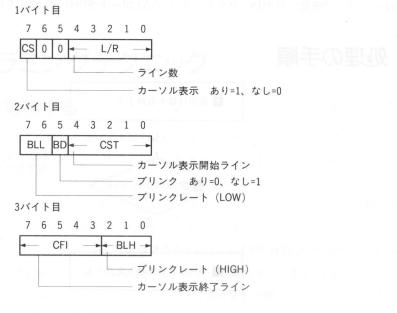



Figure 1　CSRFORMコマンドのパラメータ

ライン数は、**1** で取得した画面表示行数を元にして適切に設定しなければならない。これが正しく設定されないと、テキスト画面の表示が正しくならない。

CSRFORM コマンドのパラメータの構成を見ればわかるように、CSRFORM コマンドを使用すればカーソルの ON や OFF、ブリンクの有無、ブリンクの速さ、カーソルの大きさなどを変更することができる。

実際にテキスト用の GDC に発行する CSRFORM コマンドのパラメータは、List 3 のように準備する。

List 3 CSRFORM コマンドのパラメータ用のデータ

```
cdata   db      93h,00h,9bh       ; 20 行, ボックス
        db      8fh,00h,7bh       ; 25 行, ボックス
        db      93h,10h,8bh       ; 20 行, アンダーライン（太）
        db      8fh,0eh,7bh       ; 25 行, アンダーライン（太）
        db      93h,11h,8bh       ; 20 行, アンダーライン（細）
        db      8fh,0fh,7bh       ; 25 行, アンダーライン（細）
```

CSRFORM コマンドを発行するプログラム例を List 4 に示す。このプログラムは 200 ライン表示のディスプレイを使用するモードや、ハイレゾモードには対応していない。ハイレゾモードや 200 ライン表示のモードに対応するためには、それぞれのデータを用意しなければならない。

List 4 CSRFORM コマンドの実行例

```
; AX: 0=ボックス（通常の状態）
;     1=太いアンダーライン
;     2=細いアンダーライン
        mov     cl,6            ; 1 つの形状指示で 6 バイトのデータを使う
        mov     bx,ax
;
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        mov     al,4bh          ; CSRFORM コマンド
        out     62h,al          ; GDC にコマンド出力
        mov     cx,3            ; GDC に 3 バイトのデータを出力
gdcdata:
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        mov     al,cdata[bx]    ; カーソル形状データ
        inc     bx
        out     60h,al          ; カーソル形状データを GDC に設定
        loop    gdcdata
```

■GDCの設定値の保存

GDCに設定したパラメータは読み出しができないので、一部分だけのパラメータの変更はできない。このため、プログラムでカーソル形状などを変更しても、CRT BIOSあるいはMS-DOSのエスケープシーケンスなどでカーソルのONやOFFに関する処理を行うと、カーソルのONやOFFをGDCに設定するときにカーソル形状が再設定されてしまう。

これを防ぐためには、CRT BIOSのカーソルのONやOFFの処理を独自の処理に置き換えるか、CRT BIOSが呼び出された直後にもう一度カーソル形状を設定する必要がある。

■表示行数を取得する際の注意点

現在のMS-DOSでは表示行数の設定にCRT BIOSを使用しているため、CRT状態フラグと実際の表示行数は一致している。しかし、古いバージョンのMS-DOSの中には、表示行数の設定を直接GDCで制御しているため、CRT状態フラグと現在の画面表示が一致しないことがある。

MS-DOSで管理している表示行数の情報とCRT BIOSの管理している情報が一致していないと、CRT BIOSでカーソルの制御を行ったときに画面表示が乱れることがある。したがって、場合によってはMS-DOSのワークエリア0060:0112Hを参照する方がよい場合もある。

ライブラリ

SetCursorForm

解説　カーソルの形状をテキスト用のGDCに設定する。書式はつぎのとおり。

void **SetCursorForm**(int *form*)

form　カーソルの形状を指定する

値	カーソルの形状
0	ボックス（通常の状態）
1	太いアンダーライン
2	細いアンダーライン

この効果は一時的なものである。なお、200ライン表示のディスプレイを使用するモードや、ハイレゾモードには対応していない。

戻り値　なし

サンプル　CSRFORM.C

COLUMN

外字の未使用部分

188文字のユーザ定義文字が使用できる機種でも、ハードウェア的には256文字分のユーザ定義文字が用意されている。ユーザ定義文字の個数が188文字に制限されているのは、文字コードの範囲を漢字コードの範囲に合わせているためである。したがって、テキストVRAMを直接アクセスして文字を書き込む場合は、256個のユーザ定義文字を扱うことができる。

パソコンの電源を入れたとき、ユーザ定義文字の未使用部分はランダムな値になっている。この部分はシステムによって初期化されない。そのため、電源が入ったままの状態ならば、リセットスイッチでシステムを再起動してもユーザ定義文字の未使用部分は書き替わらない。そこで、ユーザ定義文字の未使用部分に特定のデータを書き込んでおけば、そこを参照することによって、システムが電源の投入で起動したのか、リセットスイッチで起動したのかをプログラムで判別することが可能となる。ただし、ユーザ定義文字の全エリアを書き換えてしまうようなプログラムを動かしたような場合はこの限りではない。

また、プリンタポートに電源の入ったプリンタを接続していると、パソコン本体の電源を切っても、わずかな電力がプリンタポートを伝ってユーザ定義文字を記憶しているRAMに供給されている。そのため、パソコンの電源をOFFにしていてもユーザ定義文字の未使用領域のデータが消えない。これは、RAMの性能が上がって非常にわずかな電圧と電流でデータを保持できるようになったためである。ユーザ定義文字の未使用部分は、約2Kバイトほどの容量がある。プリンタポートにプリンタを接続してわずかな電力を供給しただけで、2Kバイトの不揮発性メモリができ上がってしまうのである。

テキスト画面の表示を止める高速な外字登録

ユーザ定義文字（いわゆる外字）はCRT BIOSを使用して登録することができるが、登録処理に時間がかかる。とくに多くのユーザ定義文字を登録する場合は、処理に数秒もかかってしまう。

CRT BIOSのユーザ定義文字の登録処理が遅いのは、KCG（漢字キャラクタジェネレータ）がコードアクセスモードになっているためである。CRT BIOSを使用してKCGをドットアクセスモードに変更すれば、登録処理は大幅に短縮できる。

そのため、ここではアプリケーションプログラム起動時などで多くのユーザ定義文字を登録しなければならないときに、これを高速に行うための方法を紹介する。ただし、KCGをドットアクセスモードにするとテキスト画面の全角文字の部分が文字化けしてしまうので、画面表示を一時的に停止させている。画面表示をしながら高速に外字登録をするには、別節の「テキスト画面の表示を止めない高速な外字登録」を参照してもらいたい。

▶ポイント

- CRT BIOSを使用してKCGをドットアクセスモードに変更すれば、CRT BIOSのユーザ定義文字の設定処理はVSYNC割り込みを使用しないので、ユーザ定義文字の書き込みも瞬時に終わる。
- KCGをドットアクセスモードにすると画面上の全角文字の表示がANK文字に化けてしまうので、一瞬だけ画面表示をOFFにして文字化けを見せないようにする。
- CRT BIOSを使用してテキスト画面の表示や停止すると、一瞬だが画面表示が乱れる機種がある。そこで、VSYNC期間中にCRT BIOSを呼んで回避する。
- CRT BIOSのユーザ定義文字の登録ではフォントバッファの先頭に2バイトのワークエリアが必要なので、専用のフォントバッファにフォントデータを転送してから登録する。

▶プログラミングテクニック

1 テキスト画面の表示を停止する

KCGをドットアクセスモードに変更すると、テキスト画面上の全角文字は文字化けして表示されてしまう。この文字化けを見せないために、テキスト画面の表示をユーザー定義文字の登録が終わるまで停止しておく。

テキスト画面の表示を停止するには、CRT BIOSのファンクション0DHを使用するのが簡単である。ただし、CRT BIOSを使用してテキスト画面の表示を停止すると、一瞬だが画面の表示が乱れる機種がある。これを防ぐためには、VSYNCが1になるのを待って、VSYNC期間中にCRT BIOSを呼べばよい。VSYNCを待つプログラムは「テキストVRAMの基礎知識」で紹介したvsync_waitルーチンを利用する。プログラムをList 1に示す。

List 1　テキスト画面の表示停止

```
call    vsync_wait        ; 画面の乱れを防止するため VSYNC を待つ
mov     ah,0dh            ; テキスト画面表示停止
int     18h
```

処理の手順

1 テキスト画面の表示を停止する
↓
2 KCGをドットアクセスモードに変更する
↓
3 ユーザ定義文字を書き込む
↓
4 KCGをコードアクセスモードに変更する
↓
5 テキスト画面の表示を開始する

2 KCGをドットアクセスモードに変更する

ユーザ定義文字の登録や読み出しはCRT BIOSのファンクション1AHを使って行えるが、通常はKCGがコードアクセスモードになっているため、1文字の登録で1回のVSYNC割り込みを必要とする。VSYNC割り込みは1秒間に約60回しか発生しないため、多くの文字を登録する場合は処理に時間がかかる。

そこで、CRT BIOSのファンクション1BHを呼び出してKCGをドットアクセスモードに変更しておく。このとき、CRT BIOSはシステム共通域のCRT状態フラグ0000：053CHのKCGアクセスモードフラグ（bit3）を1にセットする。

CRT状態フラグのKCGアクセスモードフラグが1になっていれば、CRT BIOSのユーザ定義文字の設定処理はVSYNC割り込みを使用しなくなるので、ユーザ定義文字の書き込みは瞬時に終わる。プログラムをList 2に示す。

List 2　KCGをドットアクセスモードに変更

```
mov     ax,1b01h            ; KCGドットアクセスモードにする
int     18h
```

CRT BIOSのユーザ定義文字の登録処理を高速化するためには、かならずCRT BIOSを使用してKCGをドットアクセスモードにする必要がある。I/Oポートを直接操作してKCGドットアクセスモードにしても、CRT状態フラグのKCGアクセスモードフラグが0のままだと、CRT BIOSのユーザ定義文字の設定処理はVSYNC割り込みを使用してしまう。

3 ユーザ定義文字を書き込む

ユーザ定義文字のデータは1文字につき32バイトを使用するが、CRT BIOSを使用してユーザ定義文字を登録する場合は、フォントデータの前に2バイトのワークエリアが必要である。

フォントデータは使用する文字数分を連続して用意しておくのが一般的なので、登録処理のためだけにフォントデータの前に2バイトのワークエリアを設けることはできない。そこで、ユーザ定義文字の登録処理を行う関数内でワークエリア付きのフォントバッファを別に設けておき、一度フォントデータをバッファに転送してから登録する。このバッファは2バイトのワークエリアと32バイトのフォントデータを格納するため、34バイトの大きさが必要である。

CRT BIOSでユーザ定義文字を登録する場合は、INT 18Hのファンクション1AHを使用する。このファンクションでは、BXレジスタにバッファの先頭アドレスのセグメントを、CXレジスタにオフセットを入れ、DXレジスタに登録コードを入れる。プログラム例をList 3に示す。

List 3　ユーザー定義文字の登録

```
; SI フォントデータの先頭アドレス
; DX 登録コード
        .DATA
fbuf    db      2+32 dup (?)        ; フォントバッファ

        .CODE
        mov     ax,ds
        mov     es,ax
        mov     di,OFFSET DGROUP:fbuf+2
        mov     cx,32
        cld
        rep movsb                   ; フォントデータを転送
;
        mov     bx,ds
        mov     cx,OFFSET DGROUP:fbuf
        mov     ah,1ah              ; ユーザ定義文字の書き込み
        int     18h
```

登録コードには、PC-9801E/F/M/U2では7621H〜765FHの63文字が、他の機種(PC-9801初代を除く)では7621H〜767EHおよび7721H〜777EHの188文字が指定できる。ユーザ定義文字はハードウェア的にはPC-9801E/F/M/U2では64文字、他の機種(PC-9801初代を除く)では256文字あるが、登録コードをJIS漢字コードの範囲内にするため、利用範囲が制限されている。

実際は複数のユーザ定義文字を登録するので、登録処理を必要回数だけ繰り返すことになる。

4 KCGをコードアクセスモードに変更する

ユーザ定義文字の書き込みが終わったら、KCGを通常のコードアクセスモードに戻す。これを行わないと、画面上の全角文字が文字化けしてしまう。プログラムをList 4に示す。

List 4　KCGをコードアクセスモードに変更

```
mov     ax,1b00h        ; KCG コードアクセスモードにする
int     18h
```

5 テキスト画面の表示を開始する

最後に、画面上の全角文字の文字化けを見せないために停止していた表示を再開する。停止のときと同じように、CRT BIOSを利用すると一瞬画面が乱れるので、VSYNCを待って、VSYNC

期間中にCRT BIOSを呼ぶ。もちろん、コードアクセスモードへの変更の処理を怠ると、画面上の全角文字は文字化けしてしまう。プログラムをList 5に示す。

List 5　テキスト画面の表示開始

```
call    vsync_wait        ; 画面の乱れを防止するため VSYNC を待つ
mov     ah,0ch            ; テキスト画面表示開始
int     18h
```

▶ワンポイント

■グラフBIOSとCRT BIOS

グラフBIOSのグラフィック画面の表示開始処理では、CRT BIOSと異なりBIOS内でVSYNCが1になるのを待ってから画面表示をONにしている。そのため、グラフBIOSでグラフィック画面の表示を開始する場合は、VSYNCを待つような処理は不要である。

ライブラリ

SetUcgBios

解説　複数のユーザ定義文字を登録する。登録作業中は画面表示がOFFになる。

void **SetUcgBios**(int *chrcode*, char * *fontbuf*, int *num*)

chrcode　登録するユーザ定義文字の最初の登録コード
fontbuf　ユーザ定義文字のフォントデータの先頭アドレス
num　登録するユーザ定義文字の個数

戻り値　なし

サンプル　LOADUCG.C

COLUMN

CRT BIOSの設計

PC-9800シリーズのBIOSには不備が多い。たとえば、通常、画面表示に必要と思われる1文字表示のような機能がCRT BIOSに存在しない。

こうしたCRT BIOSの機能不足のためか、初期のMS-DOSのIO.SYSは、CRT BIOSを使用せず、独自にGDCを操作していた。しかし、MS-DOSの管理情報とCRT BIOSの管理情報に矛盾が生じる場合があり、アプリケーションがMS-DOSのエスケープシーケンスとCRT BIOSをともに使用すると画面が乱れることがあった。このため、現在のMS-DOSのIO.SYSでは独自にGDCを操作するのをやめ、CRT BIOSで間に合う部分はCRT BIOSを使用するように変更されている。

PC-9800シリーズではGDCの機能によりカーソルを表示しており、GDCの設定を変えることによりカーソルの大きさや点滅の有無、点滅の速さが変更できる。しかし、この設定をあとでGDCから読み出すことはできない。このような設定が読み出せないデバイスを使用する場合、通常はBIOSがそれらのデータを管理し、特定のメモリエリアに設定を記録しておき後から参照が可能なようにシステムを設計する。ところが、PC-9800シリーズではこの役目を担うはずのCRT BIOSがGDCに関する管理はほとんど何もしていない。そのため、カーソルの大きさや点滅速度などを変更したいアプリケーションは、それぞれが独自にGDCを操作することになってしまった。

カーソル位置などもBIOSが管理していればよかったのだが、PC-9800シリーズにはカーソル位置を管理するBIOSが存在しない。そのため、日本語入力FEPなどでは、カーソル位置を取得するためにMS-DOSのワークエリアを直接参照したり、GDCからカーソル位置を読み出すなど、特殊な手法を使わざるを得なくなってしまっている。

テキスト画面の表示を止めない高速な外字登録

ユーザ定義文字の登録には、CRT BIOS による方法と I/O ポートによる方法がある。KCG をドットアクセスモードに変更すれば CRT BIOS を使用しても高速にユーザ定義文字を登録することができるが、テキスト画面の全角文字が化けしてしまう。したがって、テキスト画面の表示を停止できない状況では、CRT BIOS の外字登録をコードアクセスモードで使用するしかない。だが、コードアクセスモードの登録処理は非常に遅い。

しかし、I/O ポートを直接操作することにより、実行速度が十分に速い機種では、テキスト画面の表示を変えずに登録処理を数倍速く行う方法がある。ここでは、この I/O ポートを直接操作してユーザ定義文字を登録する方法を解説する。

▶ポイント

- ユーザ定義文字の登録は CRT のブランク区間で行えば、ドットアクセスモードでも画面表示が乱れることはない。
- 1 回の VSYNC の期間に連続して何文字も登録すれば、1 文字づつ VSYNC を待つ CRT BIOS よりも高速に処理できる。

▶プログラミングテクニック

1 VSYNC を待つ

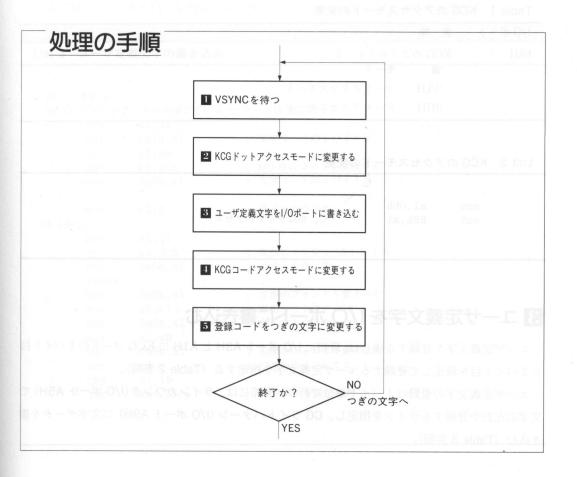
VSYNC が 1 であることを確認する場合、一時的に割り込みを禁止して VSYNC の状態をチェックする。VSYNC が 1 でなければいったん割り込みを許可して VSYNC を待ち、その後 VSYNC のチェックを行う直前に再度割り込みを禁止する。これを行わないと、長期間割り込みを禁止することになり、他の割り込みを使ったプログラムを止めてしまうことになる。

このプログラムは、基礎知識で紹介した方法と同じだが、後のプログラムと合わせるために若干修正したので List 1 に示しておく。

List 1　VSYNCの調査

```
        cld                         ; あとの lodsw 命令のため
        jmp     short sync_main

sync_wait:
        popf
        jmp     $+2                 ; リカバリタイム兼他の割り込み処理用
        jmp     $+2

sync_main:
        pushf
        cli
        in      al,060h             ; GDC のステータスを読む
        test    al,20h              ; VSYNC か
        jz      sync_wait
```

2 KCGドットアクセスモードに変更する

KCGがコードアクセスモードになっている場合は、ユーザ定義文字の書き込みはVSYNCが1の状態のときのみ可能である。しかし、VSYNCが1になるのを待ったとしても、ユーザー定義文字を書き込んでいる間にVSYNCが0になる可能性がある。

KCGをドットアクセスモードにすればユーザ定義文字の書き込みは常時行える。そのため、書き込みの前にKCGをドットアクセスモードにしておき、VSYNCが1の期間中にユーザ定義文字の書き込みが終わらなかった場合でも書き込みが失敗しないようにする。

VSYNCが1の期間に書き込みが終わらずVSYNCが0になってしまっても、VSYNCよりも画面表示のブランク区間の方が長いので、ブランク期間中に書き込みは終了する。そして、書き込み終了後にKCGをコードアクセスモードに戻せばテキスト画面の全角文字が化けることはない。

KCGのアクセスモードは**I/Oポート68H**に出力する値によって制御できる。0BHを出力すればドットアクセスモードに、0AHを出力すればコードアクセスモードになる（Table 1参照）。

KCGをドットアクセスモードに切り替えるプログラムをList 2に示す。

Table 1　KCGのアクセスモードの変更

I/Oポート	意味
68H	KCGのアクセスモード 値　モード 0AH　コードアクセスモード 0BH　ドットアクセスモード

List 2　KCGのアクセスモードの変更

```
        mov     al,0bh
        out     68h,al              ; KCGドットアクセス
```

3 ユーザ定義文字をI/Oポートに書き込む

ユーザ定義文字を登録する場合は、最初に**I/OポートA3H**と**A1H**にKCGコードの1バイト目と2バイト目を設定して登録するユーザ定義文字を指定する（Table 2参照）。

ユーザ定義文字の登録は1バイト単位で行う。実際には、**ラインカウンタ（I/OポートA5H）**で文字の左右や登録するラインを指定し、**CGライトパターン（I/OポートA9H）**に文字データを書き込む（Table 3参照）。

Table 2　ユーザ定義文字のコード指定

I/O ポート	意　味
A3H	指定する外字の JIS 漢字コードの上位バイトから 20H 引いた値
A1H	指定する外字の JIS 漢字コードの下位バイト

Table 3　ユーザ定義文字の書き込み

I/O ポート	意　味	
A5H	I/O ポート A9H に書き込むビットパターンの位置	
	ビット	意　味
	bit0〜bit3	ビットパターン上からの位置
	bit4	常に 0
	bit5	左半分（1）か右半分（0）かの指定
	bit6〜bit7	常に 0
A9H	A5H で指定した位置のビットパターン	

ラインカウンタ(I/O ポート A5H) は bit5 が 1 ならば左側、0 ならば右側の文字の指定で、bit0〜bit3 ではライン（0〜15）を指定する。残りのビットは 0 にしておく(List 3 参照)。

List 3　ユーザ定義文字の書き込み

```
; DX    登録コード
; DS:SI フォントデータの先頭アドレス（1 データは 32 バイト）
        mov     ax,dx
        out     0a1h,al         ; 文字コードの 2 バイト目
        mov     al,ah
        sub     al,20h          ; KCG 用にコードを変換する
        out     0a3h,al         ; 文字コードの 1 バイト目

        mov     cl,0            ; ラインカウンタ
set_loop:
        mov     al,cl
        or      al,20h          ; 左右ビットを 1（左）にする
        out     0a5h,al         ; ラインカウンタ
        lodsw
        out     0a9h,al         ; 左側のフォントを書き込む
        mov     al,cl           ; 左右ビットは 0（右）
        out     0a5h,al         ; ラインカウンタ
        mov     al,ah
        out     0a9h,al         ; 右側のフォントを書き込む
        inc     cl              ; つぎのラインにする
        cmp     cl,16
        jnz     set_loop        ; 16 ラインを繰り返す
```

4 KCG コードアクセスモードに変更する

ふたたびKCGをコードアクセスモードに戻す。KCGコードアクセスモードに変更するには、I/Oポート68Hに0AHを出力する。プログラムはList 2の出力する値を0AHに変更すればよい。

5 登録コードをつぎの文字に変更する

ユーザ定義文字を188文字まで使える機種では、登録コードは連続しない2つの区にわかれている。そこで、連続してユーザ定義文字を登録する場合は、つぎに登録する文字コードが別の区の文字かどうかを判断し、区を移動した場合はその処理をしなければならない。プログラム例をList 5に示す。

なお、本関数では、VSYNCが0になったあとのブランク期間もユーザ定義文字の設定に利用している。

List 5　登録コードをつぎの文字に変更

```
; DX  登録コード
        inc     dl              ; つぎの登録コードへ
        cmp     dl,7fh          ; 1区の終わりか
        jnz     skip
        mov     dl,21h
        inc     dh              ; 1区上の登録コードへ
skip:
```

ライブラリ

SetUcgIo

解説　複数のユーザ定義文字を登録する。書式はつぎのとおり。

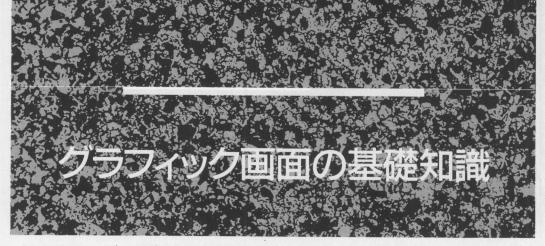
void **SetUcgIo**(int *chrcode*, char * *fontbuf*, int *num*)

chrcode　登録するユーザ定義文字の最初の登録コード。
fontbuf　1ユーザ定義文字のフォントデータの先頭アドレス。
num　登録するユーザ定義文字の個数。

戻り値　なし

サンプル　LOADUCG.C

グラフィック基礎編

グラフィック画面の基礎知識

PC-9800シリーズは、互換性はある程度保っているものの、グラフィックのハードウェアは初代のものと最近のもとのとでは大きな違いがある。ここでは、グラフィック機能の解説の前に現在までのPC-9801のグラフィックに関するハードウェアの差異による分類を行い、標準的なグラフィックのハードウェア構成を解説する。

PC-9800シリーズのグラフィックの変遷

PC-9801が最初に発表されたとき、その640×400ドット8色のグラフィックはかなり高機能なものと思われた。その当時、IBM-PCのCGAディスプレイボードは、640×200ドット2色であった。

そのあとPC-9801E/F/Mが発表され、我が国における標準のグラフィック仕様を確立した。このハードは高機能なCRTのコントローラであるGDCを2個搭載し、32Kバイトのグラフィック VRAMにより構成される640×400ドット(1バイトで8ドット)のグラフィックプレーンを3枚持ち(2^3で8色表示可能)、さらにそのグラフィック画面を2画面持っていた。しかし、この巨大な96Kバイト×2画面のグラフィック VRAMはCPUからの書き込みや読み出しが非常に遅く、さらにグラフィックにドットデータを書き込むとき、3プレーンそれぞれに書き込む必要があるため、グラフィックの描画速度はかなり遅いものであった。

そこでPC-9801VMが登場した。32Kバイトのグラフィックプレーンをさらに1枚追加し、パレット機能を強化することにより、4096色中16色表示(ただし、PC-9801VM2/VM0/VM4/VF2では16色ボードはオプション)が可能になった(数か月前に発表のPC-9801U2でも実現していた)。さらに、GRCG (Graphic Charger)による4プレーン同時書き込み、読み出しもできるようになり、グラフィックの機能は大きくアップした。

そしてPC-9801UV2以降のマシンからグラフィック VRAMがデュアルポートRAMとなって、CPUからのアクセス速度も改善された。さらに、PC-9801VX2以降のCPUが80286、80386のマシンではGRCGと互換性を保ったまま拡張強化したEGC (Enhanced Graphic Charger)を搭載し、現在に至っている。

PC-9800シリーズのグラフィックハードウェアの分類

PC-9800シリーズのグラフィック部分のハードウェアを分類すると、Table 1のようになる。

Table 1　PC-9800シリーズの主要なグラフィック構成

タイプ	解像度	色　数	画面数	その他
初期タイプ	640×400ドット	8色	2画面	PC-9801初代では1画面
VMタイプ	640×400ドット	4096色中16または8色	2画面	GRCG搭載
EGCタイプ	640×400ドット	4096色中16色	2画面	EGC、CGウィンドウ搭載
LTタイプ	640×400ドット	モノクロ	1画面	
ハイレゾタイプ	1120×750ドット	4096色中16色		

それぞれのタイプ名は独自に命名したものであり、以下のグラフィックの解説で使用するものである。これ以外に、PC-H98シリーズにおけるAGDC、E²GC、16万色中256色同時表示のグラフィックハードウェアがあるが、それを使っているソウトウェアは少ない現状なので、ここでは扱わないことにする。各タイプの詳細は、後で述べる。

以降のグラフィックの説明では、グラフィックの初期化、パレット操作、GDCについては初期タイプを、VMタイプ、EGCタイプを対象とし、GRCGについてはVMタイプとEGCタイプを、EGCについてはEGCタイプを対象とする。LTタイプやハイレゾタイプはPC-9800シリーズの中では特殊であり本書では省略する。

初期タイプ

このグラフィックハードウェアはPC-9801F/Mのもので、PC-9801のグラフィックハードウェアの基本といえる。解像度は640×400ドットである。32Kバイトのグラフィックプレーンを3枚持ち、デジタル8色（黒、青、赤、マゼンタ、緑、シアン、黄色、白）の表示ができる。

Figure 1のメモリマップに示すように3枚のグラフィックプレーンがCPUのメモリマップ上に連続的に配置されているため、CPUからのアクセスは容易である。また同じメモリアドレス上にもう一枚のグラフィック画面を持ち、I/Oポートで制御することにより、表示画面および描画画面を切り替えることができる。

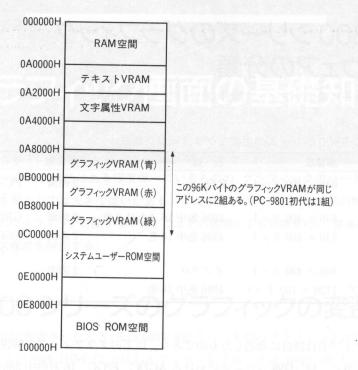



Figure 1　初期タイプのグラフィック VRAM

このタイプに属する PC-9801 は、つぎのとおりである。

- PC-9801E
- PC-9801F1/2/3
- PC-9801M2/3

PC-9801 初代は、このタイプとほぼ共通だが、グラフィック画面が 1 画面のみである。

VM タイプ

PC-9801 のグラフィック機能は PC-9801VM2 の登場で大きく変化した(実際には、PC-9801U2 からその片鱗はあった)。事実上、このグラフィック機能が PC-9801 の標準と考えてよいであろう。

640×400 ドットの解像度は初期タイプと同様だが、グラフィックプレーンを 1 枚追加して、4096 色中 16 色を表示できるようになっていた。また、GRCG を搭載した。

ゲームではこのハードが必須のものが多い。なぜなら GRCG を使った 4 プレーン同時書き込みによる高速描画と 4096 色中 16 色表示の機能が、画面表現を重んじるゲームでは有効なものであるからである。

ソフトウェアからは直接影響がないが、このタイプもデュアルポート RAM を搭載したマシン

と、そうではないマシンに分けられる。デュアルポートRAMを搭載した機種では、グラフィックVRAMへのアクセスが高速になっている。このタイプのメモリマップをFigure 2に示す。

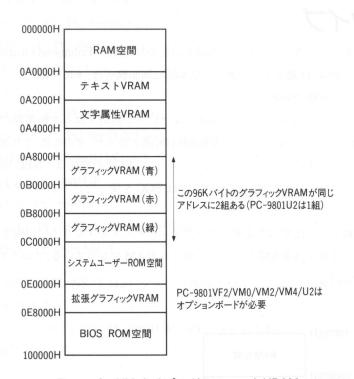



Figure 2　VMタイプのグラフィックVRAM

VMタイプのデュアルポートRAMのないPC-9801は、つぎのとおりである。

- PC-9801VF、PC-9801VM0/2/4

また、デュアルポートRAMを搭載している機種はつぎのとおりである。

- PC-9801UV2、PC-9801VM21
- PC-9801UV21、PC-9801LV21
- PC-9801CV21、PC-9801UV11
- PC-9801VM11、PC-9801LV22
- PC-9801DO、PC-9801N

CPUがV30（V30HLは除く）のみの機種は、ほとんどこの分類に属すると考えて良い。PC-9801VF2/VM0/VM2/VM4では16色ボードがオプションで、これを加えて初めて4096色中16色の表示が可能になる。このボードがないときは4096色中8色となる。PC-9801U2では、16色

ボードがオプションであり、このボードを付設するとグラフィック画面が1画面であることを除いて、この分類のマシンと共通となる。

EGCタイプ

VMタイプに加えて、さらにスピードの向上のためにEGC（Enhanced Graphic Charger）を搭載したのがこのEGC搭載タイプである。基本的な画面構成は、640×400ドット、4096色中16色と、VMタイプと同様である。

EGCを搭載したことにより、グラフィックの4プレーン同時ブロック転送等の機能が加わり、GDCによる描画においても4プレーン同時描画が可能となった。そのため、より高速なグラフィック描画を行うことができるようになった。ただし、このEGCの機能は、NEC独自のもので、一般には公開されていないため、一般のプログラマはEGCを使ったソフトウェアがなかなか作成できない。

また、EGC搭載マシンはすべて文字のグラフィックデータをメモリ上に出現させるCGウィンドウの機能も備えており、VRAMがデュアルポートRAMになっている。このタイプのメモリマップをFigure 3に示す。

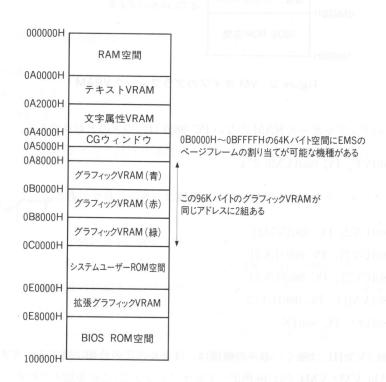



Figure 3　EGCタイプのグラフィックVRAM

このタイプに属するPC-9801は、つぎのとおりである。

- PC-9801VX0/2/4、PC-98XL model 1/2/4（ノーマルモード）
- PC-9801VX01/21/41、PC-98XL²（ノーマルモード）
- PC-9801UX21/41、PC-9801RA2/5
- PC-9801RX2/4、PC-9801LS2
- PC-98RL model 2/5/21/51（ノーマルモード）
- PC-9801EX2/4
- PC-9801RA21/51、PC-9801RS21/51
- PC-9801RX21/51、PC-9801DX2/5
- PC-9801DS2/5、PC-9801DA2/5/7
- PC-9801UR、PC-9801UF
- PC-9801NS、PC-9801DO＋
- PC-9801NV、PC-9801NS/E、PC-9801NC
- PC-9801FA、PC-9801NS/T

PC-9801NV、NS/E、NC、NS/Tは、レジューム機能をONにすると、VMタイプのグラフィックハードとして動作する。

『テクニカルデータブック』ではPC-9801UR、UF、NVはGRCGのみとなっているが、実際にはEGCが搭載されている。

LTタイプ/ハイレゾリューションタイプ

本書のグラフィックス編では扱わないが、PC-9800シリーズのグラフィックハードウェアとしてLTタイプとハイレゾリューションタイプの概要を説明する。

PC-9801LT/HAは、グラフィックプレーンが1枚で640×400ドット32KバイトのグラフィックVRAMしか持たず、テキストVRAMもない。したがって、表示できる色は黒と白のみで、テキストもグラフィックで表示する。したがってグラフィックやテキストVRAMを直接操作しているソフトウェア(PC-98LT/HA専用のものを除く)は、ほとんど動作しない。このタイプのメモリマップをFigure 4に示す。このタイプに属するPC-9801は、つぎのとおりである。

- PC-98LT model 1/2
- PC-98LT model 11/21
- PC-98HA

PC-98XAで登場したのが、ハイレゾリューションと呼ばれるグラフィックのハードウェアである（本書ではハイレゾと略す）。このタイプは、1120×750ドットの高解像度のグラフィック画面を持つ。128Kバイトのプレーン4枚でグラフィック画面を構成し4096色中16色の表示が可能である。

通常のPC-9800シリーズのグラフィックハードと大きく異なるため、グラフィックやテキストVRAMを直接アクセスするノーマルPC-9801用のソフトウェアは動作しない。ただし、テキストVRAMはアドレスが異なるのみで構造は同じであるため、テキストのみを扱うソフトウェアでは、ハイレゾかどうかを判断して動作するソフトウェアも多くなってきている。

このタイプのメモリマップをFigure 4に示す。このタイプに属するPC-9801は、つぎのとおりである。

- PC-98XA model 1/2/3
- PC-98XL model 1/2/4（ハイレゾモード）
- PC-98XL²（ハイレゾモード）
- PC-98RL model 2/5（ハイレゾモード）
- PC-98RL model 21/51（ハイレゾモード）

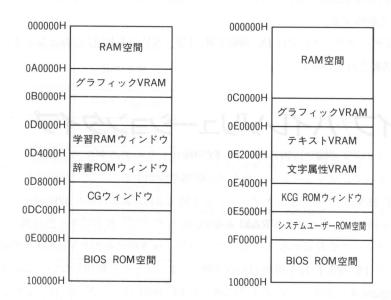



Figure 4　LTタイプ/ハイレゾタイプのグラフィックVRAM

CPU/GDC/GRCG/EGCの関係

ここまで、各タイプのもつプレーン数や、グラフィックチャージャの名称などを紹介してきたが、CPUからこれらはどのように見えるのかを解説する。まず、Figure 5にグラフィックハードウェア周りの関係を示す。

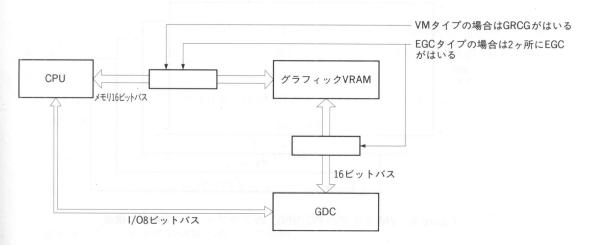



Figure 5　CPU、GDC、グラフィックVRAMの関係

基本的に、グラフィックVRAMに描画するには2つの方法が用意されている。1つはCPUからメモリバスを通して普通のメモリと同じように描画する方法である。もう1つは、GDCにコマンドを送って描画する方法である。GDCは、後で紹介するが直線や円を描画するコマンドをもっており、当初はCPUで描画するより高速であった。しかし、CPUが高速になった現在、Figure 5を見てもらえばわかるように、CPUは16ビットバスを通じてテキストVRAMにアクセスできるが、GDCへのコマンドは8ビットのI/Oバスを通して送られるのでかえってGDCで描画する方が遅くなるといった面さえある。

このようにグラフィックVRAMは、CPUとGDCによるアクセスが可能であるため、両者の同時アクセスが生じないようプログラムを組む必要がある。

VMタイプやEGCタイプに、搭載されているGRCGやEGCは、Figure 5のようにCPUとグラフィックVRAM、あるいはGDCとグラフィックVRAMの間にはいり、CPUやGDCからは透過的に見える。GRCGやEGCが図形を直接描画するわけではない。通常GRCGやEGCは動作しておらず従来のグラフィックVRAMへのアクセスやGDCを使った描画が行える。GRCGやEGCが働いても、CPUやGDCは通常の描画を続けるだけである。しかし、GRCGやEGCの機能により、CPUやGDCが1プレーンに描画するだけで、自動的に4プレーン同時描画できたり、どのようなデータで描画してもパターンで描画したりできるようになる。

グラフィックVRAMの構成

典型的な例として、VMタイプのPC-9801のグラフィックVRAMの構成をFigure 6に示す。

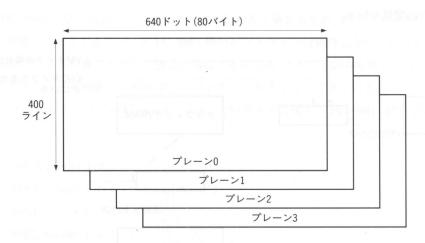



Figure 6　VMタイプのPC-9801のグラフィックVRAMの構成

プレーン1枚で32000バイト(640ドット×400ライン÷8ビット)のサイズを持ち、それが4枚で1画面を構成する。各プレーンのアドレスはTable 2のとおりである。

Table 2　プレーンとグラフィックVRAMのアドレス

プレーン	セグメント	オフセット範囲
プレーン0	A800H	0～31999
プレーン1	B000H	0～31999
プレーン2	B800H	0～31999
プレーン3	E000H	0～31999

メモリマップを見ると1プレーンのサイズは32K(1024×32)バイトになっており、768バイト(1024×32バイト−32000バイト)が余分に存在するが、グラフィックの表示には通常は使われない。

この4プレーンで構成されるグラフィック画面が2画面ある(PC-9801初代、PC-9801U2、PC-98LTシリーズを除く)。

プレーン0にデータを書き込むと、書き込んだ場所のドットの色番号のbit0が1になる。同様に、プレーン1は色番号のbit 1に、プレーン2は色番号のbit 2、プレーン3は色番号のbit 3に対応する。たとえば、プレーン0とプレーン2の同じオフセットアドレスに同じデータを書き込むと、色番号5(2進数の0101B)でデータが表示されることになる(Table 3参照)。

Table 3　プレーンとビット

ビット番号	3	2	1	0
ビットパターン	0	1	0	1 (10 進数で 5)
プレーン番号	3	2	1	0

CPU 上の 16 ビットのデータをグラフィック VRAM に書き込むと、8086 系の CPU のメモリ格納配置と同様に上位バイトと下位バイトが入れ替わってグラフィック VRAM に格納されるので注意が必要である。

たとえば、AX レジスタに 0011110011110000B が入っているとき、つぎのようにグラフィック VRAM に書き込む。

```
mov es: [0000H] ,ax          ; es＝0A800H
```

すると、グラフィック VRAM には Table 4 のように格納される。

Table 4　グラフィック VRAM へのデータ格納

アドレス	A800:0000H	A800:0001H
データ	11110000	00111100

その結果、ディスプレイには Figure 7 のように表示される。

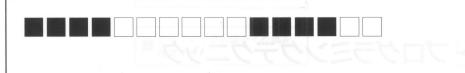



Figure 7　ディスプレイへの表示

グラフィックの初期化

PC-9801では、電源が投入された時点では、グラフィックを表示しない状態になっている。また、PC-9801のグラフィックにはさまざまなモードがあり、ただ単にグラフィックが表示されるようにするだけでは、200ラインモードになるなど問題がある。したがって、グラフィック画面を利用する前には、自分の必要とするグラフィック画面のモードで表示されるように初期化し、グラフィックの各種パラメータを適切に設定する必要がある。

ここでは、このグラフィックの初期化について解説する。

▶ポイント

- グラフィック表示を止めてから各種設定を行う。
- グラフィックBIOSを利用して、表示縦ライン数、描画画面、表示画面、カラーモードを一度に設定する。
- グラフィックハードウェアがどのタイプものかチェックしてから、引数によりデジタル8色かアナログ4096色中16色かを設定する。
- グラフィック設定データをグローバル変数に格納する。
- すべての設定が終了してからグラフィック表示を再開して初期設定を完了する。

▶プログラミングテクニック

1 グラフィック画面の表示を止める

グラフィックの初期設定を行うとき、ディスプレイにグラフィック画面を表示した状態では、画面にノイズが生じる。そこで初期設定の初めにグラフィック画面表示を停止し、初期設定が終了した時点でグラフィック画面表示を再開することでノイズを避ける。

グラフィック画面表示のON/OFFの方法には、BIOSを利用する方法と、直接I/Oポートを操作する方法がある。BIOSを利用する方法ではどのような種類のPC-9801でも確実にON/OFFができるが、処理時間や内部動作がわからないという問題がある。反面I/Oポートを利用する方法は、処理時間は短く、内部動作も明確であるが、ワークエリアの設定等がすべてのPC-9801で正

常動作するかどうか不安がある。そのため、ここで示すグラフィック初期化は、BIOSを利用することにする。

グラフィックBIOSの機能を使って表示のON/OFFを切り替えるには、**グラフィックBIOSのファンクション40Hと41Hを使う**（Table 1参照)。

Table 1　グラフィック画面の表示開始、停止

ファンクション	意　味
INT 18H 40H	グラフィック画面の表示開始 入力　なし 出力　なし
INT 18H 41H	グラフィック画面の表示停止 入力　なし 出力　なし

このBIOSでは、GDCにコマンドを出力するとともにシステム共通域の設定も行っている。たとえばグラフィックの表示状態を示すフラグは**Table 2**のようになっている。

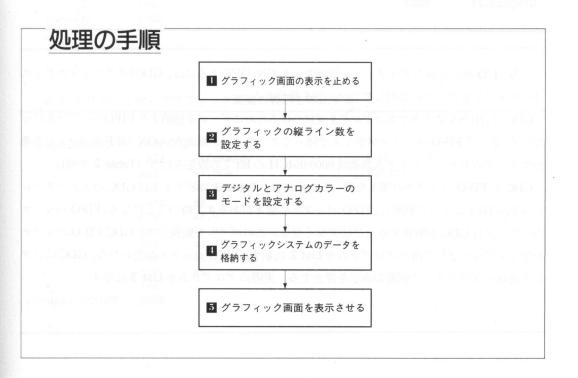



Table 2　グラフィックの状態

アドレス	意　味	
0000:054C	グラフィック画面のステータスフラグ（PRXCRT）	
	bit 7	CRT 状態
	1	表示
	0	表示停止

電源をON（またはリセット）して最初にグラフィックをONにすると、デジタルカラー、縦200ラインの設定となっている。

このグラフィックBIOSを利用する際には、スタックエリアを30バイト以上確保する必要がある。ただ、通常のアプリケーションでは30バイト以上のスタックエリアは確保してあるので、とくに前設定の必要はない。

BIOSからグラフィック画面表示を停止するプログラムをList 1に示す。

List 1　BIOSからグラフィック画面表示を停止するプログラム

```
GraphicOFF      PROC
                mov     ah,41h
                int     18h
                ret
GraphicOFF      ENDP
```

一方、I/Oポートからグラフィック画面表示をON/OFFするには、GDC(グラフィックディスプレイ・コントローラ）に対してコマンドを発行する。

GDCは、BUSYのときでもコマンドを受けつけるためにデータを格納するFIFOバッファを持っている。そこでFIFOバッファが空くまで待ってグラフィック画面表示ON/OFFのコマンドを発行する。そのあとで、システム共通域0000:054CHのbit 7の設定を行う（Table 2参照)。

GDCのFIFOバッファの空きを調べるためには、Table 3に示すようにGDCのステータスレジスタのbit 1によって判断し、FIFOバッファに空きがあるまで待つことになる。FIFOバッファ内のデータはGDCが解釈するたびに少なくなる。このポートを監視して、GDC FIFOバッファが空くまでループして待つプログラムをList 2に示す。FIFOバッファが空いたら、GDCにコマンドを送りグラフィック画面の表示を停止する。実際のプログラムをList 3に示す。

Table 3　FIFOバッファの空きをチェックするI/Oポート

I/Oポート	意　味
A0H	ステータスレジスタ ビット　意　味 *bit2*　*FIFO Empty* FIFOバッファが空であることを示す 値　意　味 0　FIFOバッファにまだコマンドがある 1　FIFOバッファになにもデータがない bit1　FIFO Full FIFOバッファがいっぱいであることを示す 値　意　味 1　FIFOバッファがいっぱい 0　FIFOバッファに空きあり

List 2　GDC FIFOバッファが空くまでループして待つ

```
FifoReady       PROC
fifo_full:      in      al,0A2h         ; GDCステータス読み込み
                test    al,02h          ; FIFOバッファがいっぱい？
                jmp     $+2             ; ウェイト
                jmp     $+2             ; ウェイト
                jne     fifo_full
                ret
FifoReady       ENDP
```

List 3　I/Oポートからグラフィック画面の表示を停止する

```
GraphicGdcOFF   PROC
                call    FifoReady                   ; FIFOバッファがいっぱい?
                mov     al,0Ch                      ; 表示OFFコマンド
                out     0A2h,al
                push    es
                mov     ax,0
                mov     es,ax
                mov     al,7Fh
                and     es:[054Ch],al
                pop     es
                ret
GraphicGdcOFF   ENDP
```

2 グラフィックの縦ライン数を設定する

PC-9801の縦ライン数は200ラインと400ラインの設定が可能である。200ラインに設定したときは、グラフィックVRAMの32Kバイトのうち上半分を使うか下半分を使うかも指定できる。

この設定には、BIOSを使う方法とI/Oポートを使う方法がある。I/Oポートを利用する方法だと多くのワークエリアの設定が必要となるため、ここではBIOSを使うことにする。

Table 4に縦ライン数を設定するグラフィックBIOSファンクションを示す。

Table 4　縦ライン数を設定するグラフィックBIOSファンクション

ファンクション	内　容
INT 18H 42H	表示領域の設定 入力　CH＝表示領域の設定値 CH　CRTモード設定値 bit 7,6　VRAM領域設定 値　意　味 01　UPPER 10　LOWER 11　ALL bit 5　カラーモード設定 値　意　味 0　カラー 1　モノクロ bit 4　表示バンク 値　意　味 0　バンク0（表） 1　バンク1（裏） 出力　なし

このBIOSファンクションでは縦のライン数の設定以外に、カラーモード設定、表示バンクの設定を行うことができる。

VRAM領域設定は、ALLで縦400ラインとなる。LOWERで縦200ライン表示開始アドレスA000:0000HでグラフィックVRAMの前半の16000バイトを使うことになり、400ラインの上半分相当になる。UPPERでは、縦200表示開始アドレスA000:3E80Hからの後半16000バイトを使うことになり400ライン時の下半分相当になる。

ここでは200ラインの設定はLOWERであり、グラフィックVRAMの前半分を利用するものとなる。実際に縦ライン数を設定するプログラムをList 4に示す。

List 4　グラフィック縦ライン数設定

```
;  int Line : 0 = 200 ライン、1 = 400 ライン

GraphicLineSet  PROC Line : WORD
                mov     ax,Line
                cmp     ax,0
                jz      lines200
                cmp     ax,1
                jz      lines400
                jmp     lines_exit

lines200:
                mov     ch,10000000b            ; LOWER COLOR BANKO
                jmp     SHORT lines_go
lines400:
                mov     ch,11000000b            ; ALL COLOR BANKO
lines_go:
                mov     ah,42h
                int     18h                     ; グラフィック BIOS
lines_exit:
                ret
GraphicLineSet  ENDP
```

I/O ポートを使って縦ライン数を変更するには、I/O ポート 68H にパラメータを出力して (09H＝200 ライン、08H＝400 ライン) モード変更を行う。しかし、このときに GDC の初期化パラメータもセットしなければならないため、かなり複雑になる。縦ラインの変更はそれほど頻繁に使用するものではないため、BIOS を使用する方法の方が望ましい。ただし、GDC の初期設定パラメータをいろいろと変更すると様々の画面（408 ライン等）を表示することが可能になる。

3 デジタルとアナログカラーのモードを設定する

グラフィック画面は、デフォルトではデジタル 8 色表示になる。

「グラフィック画面の基礎知識」で分類したなかで VM タイプ、EGC タイプのグラフィックハードウェアを持つ機種では、4096 色中 16 色のアナログ表示が可能になっている。しかし、グラフィック BIOS は初期タイプの PC-9801 を想定して作成されているため、アナログカラーモードの設定ができない（PC-9801VX21 以降のグラフ LIO では可能)。

そこで直接 I/O ポートを操作することによりカラーモードを切り替える。Table 5 に示すように、I/O ポート 6AH に 0（デジタルカラーモード）または 1（アナログカラーモード）を出力することにより、カラーモードを変更できる。プログラムを List 5 に示す。

Table 5　グラフィックモード I/O ポート

I/O ポート	Read/Write	意　味
6AH	Write	ライトモードレジスタ 値　モード 00H　デジタルカラーモード 01H　アナログカラーモード

List 5　デジタル／アナログカラーモード設定

```
;    int ColorMode : 0 = Degital、1 = Analog

GraphicColorMode PROC ColorMode : WORD
                 mov     ax,ColorMode
                 out     06Ah,al
                 ret
GraphicColorMode ENDP
```

なお、PC-9801VF/VM/U で16色ボードが装着されていないマシンは、アナログカラーモードにすると4096色中8色となる。

4 グラフィックシステムのデータを格納する

PC-9800シリーズには、PC-9801初代以来多くの種類があり、グラフィック関連のハードウェアに若干の違いがある。そのため、適切なグラフィック処理ができるように、各マシンのハードウェアをチェックする必要がある。

そこで、前もってシステム共通域を調べることにより、16色ボードの有無、GRCGの有無、EGCの有無、GDCのクロックをチェックし、グローバル変数に格納しておく。グロバール変数に代入しておくことにより、あとからこれらの情報を調べるときにセグメントを変更せずに参照できるようになる。

- GRCGの有無はシステム共通域0000:054CHのbit 1により判断できる。
- 16色ボードの有無はシステム共通域0000:054CHのbit 2により判断する。
- GDCのクロックはシステム共通域0000:54DHのbit 2により判断する。
- EGCの有無はシステム共通域0000:54DHのbit 6により判断する。PC-9801NV、NS/EではレジュームをONにするとこのビットが0になる。

これらの情報を調べるシステム共通域はTable 6のとおりである。

Table 6　グラフィックシステムデータ

アドレス	意　味	
0000:054CH	グラフィック画面ステータスフラグ（PRXCRT）	
	bit 2	拡張グラフィックVRAM（E0000H～E7FFFHのグラフィックVRAM）
	1	あり
	0	なし
	bit 1	グラフィックチャージャ
	1	あり
	0	なし
0000:054DH	グラフィックモード（PRXDUDP）	
	bit 6	EGC使用可否
	1	可
	0	不可 （NS/Eではレジューム機能を使うとEGCが使えなくなる）
	bit 2	GDCのクロック
	1	5MHz
	0	2.5MHz （GDCの実際のクロック。GDCは200ラインモードのとき必ず2.5MHzになる）

これらのシステム共通域をチェックし、グラフィックの初期化の引数も利用して、List 6に示すグローバル変数にデータを格納する。

List 6　グラフィックシステムデータ格納変数

```
_gc_set         db      0               ; GRCG
                                        ; 0 = なし、1 = あり
_egc_set        db      0               ; EGC
                                        ; 0 = なし、 1 = あり
_board16_set    db      0               ; 16色ボード
                                        ; 0 = なし、 1 = あり
_gdc_clock      db      0               ; GDCクロック
                                        ; 0 = 2.5MHz、 1 = 5MHz
_analog         db      0               ; アナログ表示
                                        ; 0 = デジタル表示、 1 = アナログ表示
_lines          db      0               ; 縦ライン数
                                        ; 0 = 200 lines 、 1 = 400 lines
```

5 グラフィック画面を表示させる

最初に画面のノイズを避けるためにグラフィック画面表示を停止した。最後に、初期設定が終了した時点でグラフィック画面表示を再開する。

BIOSからグラフィック画面を表示させるプログラムをList 7に示す。

List 7　BIOSからのグラフィック画面ON

```
GraphicON       PROC
                mov     ah,40h
                int     18h
                ret
GraphicON       ENDP
```

また、I/Oポートからグラフィック画面を表示させるプログラムをList 8に示す。

List 8　I/Oポートからのグラフィック画面ON

```
GraphicGdcON    PROC
                call    FifoReady           ; FIFOバッファが満杯?
                mov     al,0Dh              ; 表示ONコマンド
                out     0A2h,al
                push    es
                mov     ax,0
                mov     es,ax
                mov     al,80h
                or      es:[054Ch],al
                pop     es
                ret
GraphicGdcON    ENDP
```

ライブラリ

GraphicInit

解説 グラフィック画面を初期化し、グラフィック表示ON、縦ラインの設定、デジタル／アナログカラーを設定する。書式はつぎのとおり。

void **GraphicInit**(int *Lines*, int *ColorMode*)

Lines 画面の解像度の選択

ColorMode デジタル／アナログカラーの選択

引数による表示縦ライン数とデジタル／アナログカラーの設定でグラフィック画面の初期化を行う。カラー／モノクロモードはカラーモードになり、表示、描画ともグラフィック画面1になる。ここで紹介するグラフィック関数を利用する際はかならず先頭で実行しなければならない。

Lines の値	解像度
0	200
1	400

ColorMode の値	グラフィックのモード
0	デジタル表示（8色）
1	アナログ表示（4096色中16色）

戻り値 なし

サンプル GDEMO.C

パレットの設定

PC-9801は、グラフィック1画面がグラフィックプレーン3枚または4枚で構成されている。3枚のグラフィックプレーンで色を表現するデジタルカラーモードの場合、パレット操作によって、それぞれの色番号に8色のカラーコードを1つ割り当てることができる。

またVMタイプとEGCタイプのマシンでは、グラフィックプレーンが4枚になった上に、アナログカラーモードを用意し、それぞれの色番号に4096色のカラーコードを割り当てられるようになっている。これにより、かなり自由な配色でカラー表示ができるため、ゲームなどでは必須のモードとなっている。

ここでは、デジタルカラーモードとアナログカラーモードにおけるパレット操作について解説する。なお、パレット操作はグラフィックのみに対して行えるもので、テキストカラーについては適用できない。

▶ポイント

- デジタルカラーモードとアナログカラーモードがあり、パレットの設定は大きく異なる。
- 色番号はパレットレジスタ番号となり、このパレットレジスタにカラーコードを書き込むことにより、パレット設定の処理を行う。
- パレット設定の処理はVSYNC信号の発生期間に行い、グラフィック画面の乱れを防ぐ。

▶プログラミングテクニック

1 VSYNC信号の発生開始を検出する

グラフィック画面表示を乱さないようにするため、VSYNC信号の発生開始を検出し、そのあとでパレットの設定を行う。VSYNCの発生開始検出の手順は「VSYNCの検出」の節に書いたとおりである。

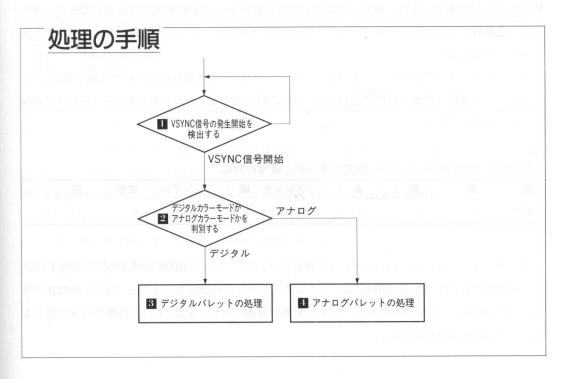
2 デジタルカラーモードかアナログカラーモードかを判別する

パレット処理は、デジタルカラーモードとアナログカラーモードとではまったく異なったものとなる。そのため、「グラフィックの初期化」の節で示した初期化関数で設定した変数（List 1 参照）を参照して、どのカラーモードかを判断する。

List 1　カラーモード判別のための変数

```
_analog         db      0               ; アナログ表示
                                        ; 0 = デジタル表示、1 = アナログ表示
```

3 デジタルパレットの処理

ディスプレイに色を表示するため、PC-9801 ではグラフィック 1 画面がグラフィックプレーン 3 枚（8 色）または 4 枚（16 色）で構成されている。3 枚のグラフィックプレーンだけで色を表現するデジタルカラーモードの場合、プレーン 0（セグメント A800H からのグラフィック VRAM）にデータ 1 を書き込むと RGB（Red、Green、Blue）の中の青の光を発することになり（プレーン 1 では赤、プレーン 2 では緑）、他のプレーンのデータが 0 であればディスプレイには青色の点が表示されることになる。また、プレーン0とプレーン1にデータ1を書き込み、プレーン2に0

を書き込めば、光の三原色の合成法則から青と赤とでマゼンタ(紫ではない) の点が表示されることになる。

このようにしてグラフィックプレーン3枚で8色を表現できる。プレーンと色の関係をTable 1に示す。

Table 1　デジタルパレットでのプレーンと色の関係

プレーン					
2	1	0	色	カラーコード	色番号
0	0	0	黒	0	0
0	0	1	青	1	1
0	1	0	赤	2	2
0	1	1	マゼンタ	3	3
1	0	0	緑	4	4
1	0	1	シアン	5	5
1	1	0	黄	6	6
1	1	1	白	7	7

このデジタルカラーモードにおけるパレット操作とは、色番号に対してカラーコードを割り当てる処理である。標準では色番号1にはカラーコード1が対応し、青が表示されるわけだが、これをカラーコード2にすることによりグラフィックプレーンのデータを変化させずに赤の表示に替えることができる。また、複数の色番号に同じカラーコードを割り当てることもできる。たとえば、色番号1と2の両方にカラーコード3を対応させると、どちらの色番号でもマゼンタを表示するようになる。

デジタルモードで用意されているパレットレジスタは0〜7と番号のふられた8個である。パレットレジスタ番号と色番号は同じ値になる。これに割り当てることができるカラーコードは、Table 2のように固定である。

Table 2　デジタルパレットに設定できる色と番号の対応

色	黒	青	赤	マゼンタ	緑	シアン	黄色	白
番号	0	1	2	3	4	5	6	7

デジタルカラーモードにおけるパレット操作には、グラフィックBIOSを使う方法と直接I/Oポートを制御する方法がある。BIOSはシステムワークエリアの設定なども行っており、信頼性が高い。一方、I/Oポートを利用するとパレット変更が高速になり、また、1つの色番号のみに対するパレットの設定も可能になる。

グラフィックBIOSからは、DS:BXに4ビットづつ8色で4バイトのデータを格納した領域の先頭アドレスを入れ、INT 18Hファンクション43Hを実行することにより、デジタルパレットの変更ができる（Table 3参照）。

Table 3　BIOSによるデジタルカラーモードのパレット操作

ファンクション	機　能				
INT 18H 43H	パレットレジスタの設定				
	入力	CS:BX	データの格納アドレス		
			アドレス	ビット	色番号
			[CS:BX]	bit 0～3	7
				bit 4～7	6
			[CS:BX + 1]	bit 0～3	5
				bit 4～7	4
			[CS:BX + 2]	bit 0～3	3
				bit 4～7	2
			[CS:BX + 3]	bit 0～3	1
				bit 4～7	0
	出力	なし			

BIOSからデジタルパレットを設定するプログラムはList 2のようになる

List 2　BIOSからのデジタルパレットの設定

```
mov     bx,ColorCode
mov     ah,43h
int     18h
```

グラフィックBIOSで行っていることとほぼ同様なことはI/Oポートを直接アクセスしても実現できる。この場合、4個のI/Oポートに、それぞれのパレットレジスタに対応するカラーコードデータを出力することにより、パレットレジスタを変更する（Table 4参照）。

1度のI/Oポートへの出力で、パレットレジスタが2つ設定できるが、同時に設定できるパレットレジスタ番号が連続していないので注意が必要である。

プログラムをList 3に示す。

Table 4　I/O ポートによるデジタルカラーモードのパレット操作

I/O ポート	Read/Write	ビット	色番号
A8H	Write	bit0～3	7
		bit4～7	3
AAH	Write	bit0～3	5
		bit4～7	1
ACH	Write	bit0～3	6
		bit4～7	2
AEH	Write	bit0～3	4
		bit4～7	0

List 3　I/O ポートによるデジタルパレットの設定

```
mov     cl,4
mov     al,[palreg0]
shl     al,cl
or      al,[palreg4]
out     0AEh,al              ; パレットレジスタ0と4設定
mov     al,[palreg1]
shl     al,cl
or      al,[palreg5]
out     0AAh,al              ; パレットレジスタ1と5設定
mov     al,[palreg2]
shl     al,cl
or      al,[palreg6]
out     0ACh,al              ; パレットレジスタ2と6設定
mov     al,[palreg3]
shl     al,cl
or      al,[palreg7]
out     0A8h,al              ; パレットレジスタ3と7設定
```

4 アナログパレットの処理

VM タイプ、EGC タイプのマシンでは、表示する色がより柔軟なものになっている。グラフィックプレーンは4枚になっただけなので、表示できる色は16色までで、色番号は0～15までである。しかしこの色番号に割り当てるカラーコードが0～4095(0～0FFFH)となり、4096色を指定できる。Table 5 に示した色番号とカーラーコードとの対応がデフォルトの状態である。

それぞれの色番号には、パレット操作によりカラーコード0～4095（0～FFFH）の色を割り当てることができる。4096色のカラーコードは緑、赤、青それぞれ4ビット、合計12ビットであり、下位4ビットが青、中位4ビットが赤そして上位4ビットが緑でそれぞれの数値で階調を表現する。この数値は0～15の範囲で大きいほど明るくなり、15でデジタルカラーモードの1と同じ明

Table 5　アナログパレットでのプレーンと色の関係

プレーン 3	2	1	0	色	カラーコード	色番号
0	0	0	0	黒	000H	0
0	0	0	1	暗青	007H	1
0	0	1	0	暗赤	070H	2
0	0	1	1	暗マゼンタ	077H	3
0	1	0	0	暗緑	700H	4
0	1	0	1	暗シアン	707H	5
0	1	1	0	暗黄	770H	6
0	1	1	1	グレー	777H	7
1	0	0	0	暗グレー	444H	8
1	0	0	1	青	00FH	9
1	0	1	0	赤	0F0H	10
1	0	1	1	マゼンタ	0FFH	11
1	1	0	0	緑	F00H	12
1	1	0	1	シアン	F0FH	13
1	1	1	0	黄	FF0H	14
1	1	1	1	白	FFFH	15

るさである。たとえばカラーコード119（077H）では青が7で半分の明るさであり、赤も半分の明るさになるため、表示される色は暗いマゼンタになる。

この、4096色中16色を表示するモードをアナログカラーモードと呼ぶ。これにより、かなり自由な配色でカラー表示ができるので、ゲームなどでは必須のモードとなっている。

アナログパレットの設定はグラフィックBIOSではサポートしていない。そのため直接I/Oポートにデータを出力して設定する。I/OポートA8Hにパレットレジスタ番号を出力したあとに、I/OポートAAH、ACH、AEHにGreen、Red、Blueのカラーコードを出力する(Table 6参照)。

Table 6　アナログカラーモード（16色モード）の操作

I/Oポート	Read/Write	ビット	値
A8H	Write	bit0～3	パレットレジスタ番号
		bit4～7	0
AAH	Write	bit0～3	Greenカラーコード
		bit4～7	0
ACH	Write	bit0～3	Redカラーコード
		bit4～7	0
AEH	Write	bit0～3	Blueカラーコード
		bit4～7	0

なお、16色拡張ボードが搭載されていない機種（PC-9801VM0/VM2/VM4/U/VF）でアナログパレット処理を行うと、I/OポートA8Hに出力されたパレットレジスタ番号の3ビット目は常に0とみなされる。そのような機種ではパレットレジスタ8〜15の設定はできず、パレットレジスタ8〜15の設定を行うと実際設定されるパレットレジスタは0〜7になってしまう。したがって、システムメモリエリア0000H:054CHのbit 2により、拡張ボードの存在をチェックしてアナログパレットの設定を行う必要がある。

パレットレジスタ1（色番号1）を操作するプログラムをList 4に示す。

List 4　パレットレジスタ1（色番号1）の設定

```
mov     dx,[color_code]  ; 設定するカラーコード
mov     al,1             ; パレットレジスタ1
out     0A8h,al          ; パレットレジスタ番号の設定
mov     ax,dx
xchg    ah,al            ; al = 最上位4ビット
out     0AAh,al          ; パレットGreen
and     ah,0F0h
mov     cl,4
shr     ah,cl
xchg    ah,al            ; al = 中位4ビット
out     0ACh,al          ; パレットRed
mov     ax,dx
and     ax,0Fh           ; al = 下位4ビット
out     0AEh,al          ; パレットBlue
```

▶ワンポイント

■パレットの読み出し

パレットレジスタの内容の読み出しは不可能なため、設定したカラーコードはプログラム中のデータエリアに保持しておくとよい。

ライブラリ

PaletteInit

解説 パレットを初期化して標準状態の色番号とカラーコードの対応にする。書式はつぎのとおり。

void **PaletteInit**(void)

デジタルカラーモードかアナログカラーモードかを判断して、Table 1、Table 5で示した標準の色番号とカラーコードの対応にする。

戻り値 なし

サンプル GDEMO.C、PAL.C

PaletteAll

解説 すべての色番号に対応するカラーコードを設定する。書式はつぎのとおり。

void **PaletteAll**(unsigned int * *ColorCode*)

ColorCode カラーコードの格納アドレス

カラーコードをデジタルカラーモードなら8ワード、アナログカラーモードなら16ワードの配列に用意し、その格納ポインタを引数で渡すことにより、1度にすべての色番号に対するパレットを設定する。

戻り値 なし

サンプル GDEMO.C

Palette

解説 指定された色番号のパレットを設定する。書式はつぎのとおり。

void **Palette**(unsigned int *ColorNo*,unsigned int *ColorCode*)

ColorNo 色番号
ColorCode カラーコード

引数で指定された色番号に、同じく引数で指定したカラーコードを設定する。他の色番号のカラーコードは変化しない。

戻り値 なし

サンプル PAL.C

VSYNC信号の検出

PC-9801ではGDCがグラフィックVRAMの内容を読み出して映像信号を出力している。GDCが読んでいるときにグラフィックの設定の変更などを行うと、グラフィック画面にノイズが発生する。そのため、設定変更は、GDCが読み出しを行っていないとき、すなわちVSYNC信号が発生している期間に行う必要がある。そのため、グラフィック関係の操作ではしばしばVSYNC信号を検出しなければならない。

なお、「テキストVRAMの基礎知識」でもVSYNCの検出法は紹介している。ここでは、VSYNC期間を十分にとるために、VSYNC信号の発生開始時点を検出する方法を解説する。

▶ポイント

- グラフィック画面を表示しながらグラフィックの設定変更を行うには映像VSYNC信号の発生期間に行う必要がある。
- I/OポートA0Hを調べてVSYNC信号を検出する。
- VSYNC信号を検出しても、VSYNC信号が終わる寸前であることも考えられるため、VSYNC信号の発生開始時点を検出する。
- VSYNC信号発生の検出を実行しているとき、VSYNC割り込みが発生すると、不安定な動作になる場合があるため、VSYNC割り込みを禁止する。

▶プログラミングテクニック

1 VSYNC(垂直同期信号)信号発生期間が終わるまで待つ

PC-9801では映像処理のハードウェアがグラフィックVRAMの内容を読み出して映像信号を出力している。

タイムチャートをFigure 1に示す。

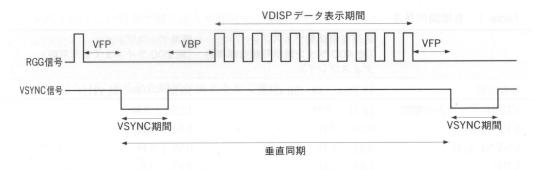



Figure 1　ディスプレイ表示のタイムチャート

VDISP データ表示期間は、映像処理のハードウェアがグラフィック VRAM の内容を読み取り、画像信号として出力している期間である。

垂直同期は、1画面の表示周期を示す。

VSYNC は1画面表示のタイミングのための信号である。この信号の発生は、GDC のステータスレジスタを読むことで検出できる。また、この信号によってハードウェア割り込み処理を実行させることもできる。

VFP＋VSYNC＋VBP の期間、GDC による表示は行われていない。そのため、画面モードの設定等はこの期間に行う。しかし、VFP の検出はむずかしいため、VSYNC 信号開始のタイミングを検出して処理を行うことになる。

それぞれの期間の長さは、ノーマルモードでは Table 1 のようになる。

グラフィックの設定を行う期間は、VS＋VBP の時間である。これは、専用高解像度ディスプレイでは 1.33 ミリ秒である。

この VSYNC 信号の発生開始時点まで待って、グラフィックの設定を行う。

VSYNC 信号を検出するには、Table 2 に示した I/O ポートのステータスレジスタ（A0H）の bit 5 をチェックする。

処理の手順

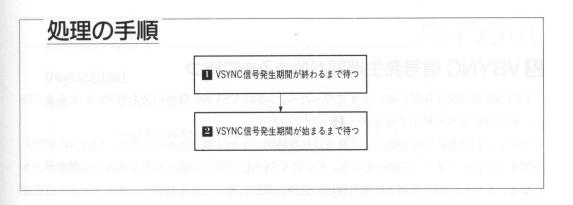



Table 1　各期間の長さ

	専用高解像度ディスプレイ（縦 400 ライン表示可能な標準的ディスプレイ）	標準ディスプレイ（縦 200 ラインまで表示可能）
垂直同期	17.72 ミリ秒（56.4Hz）	16.33 ミリ秒（61.2Hz）
VDISP データ表示期間	16.11 ミリ秒	12.52 ミリ秒
VFP	0.28 ミリ秒	0.94 ミリ秒
VSYNC 信号	0.32 ミリ秒	0.50 ミリ秒
VBP	1.01 ミリ秒	2.83 ミリ秒

Table 2　GDC I/O ポート

I/O ポート	Read/Write	意　味
A0H	Read	ステータスレジスタ
A2H	Write	コマンドレジスタ

ただし、VSYNC 信号を検出しても、VSYNC 信号が終わる寸前の時点のことも考えられる。したがって、時間がかかる処理を VSYNC 信号発生期間に行うためには、VSYNC 信号の発生開始時点を検出する必要がある。

そこで、まず VSYNC 信号を検出したら、一度 VSYNC 信号発生期間が終わるまで待つ。そして、そのあとにふたたび VSYNC 信号が検出されるまで待つ。

VSYNC 信号発生期間が終わるまで待つプログラムを List 1 に示す。

List 1　VSYNC 信号発生期間が終わるまで待つプログラム

```
vsync_ing:
            jmp     $+2             ; ウェイト
            jmp     $+2             ; ウェイト
            in      al,0A0h         ; GDC ステータスの読み込み
            test    al,020h         ; VSYNC 信号のチェック
            jnz     vsync_ing       ; VSYNC 信号が発生していたらループ
```

2 VSYNC 信号発生期間が始まるまで待つ

VSYNC 信号発生期間が終わるまで待ったら、つぎに VSYNC 信号の発生期間が始まるまで待つ。VSYNC 信号を検出する方法は 1 と同じである。

ただし、VSYNC 信号を検出した直後に処理時間のかかる割り込みが入ると、VSYNC 期間の時間をとられてしまい、問題が生じる。とくに VSYNC 信号に同期する割り込みでは問題が大きくなる。よって、この VSYNC 信号検出の処理の最初に割り込みを禁止し、グラフィックの設定

等の処理が終了した時点で割り込みを許可する必要がある。

VSYNC 信号発生期間が始まるまで待つプログラムを List 2 に示す。

List 2　VSYNC 信号発生期間が始まるまで待つ

```
            pushf
vsync_not:
            cli
            in      al,0A0h         ; GDC ステータスの読み込み
            test    al,20h          ; VSYNC 信号チェック
            jnz     vsync_go
            popf
            pushf
            jmp     vsync_not
vsync_go:
            pop     ax
```

▶ワンポイント

■割り込み禁止の解除について

VSYNC の検出では、VSYNC 割り込みを禁止しているが、その解除は具体的なグラフィック操作の方に任せている。そのまま VSYNC 割り込みを禁止しつづけると VSYNC 割り込みを使うプログラムに障害が発生するので、処理が終わったらすみやかに、割り込みを許可しなければならない。

ライブラリ

VsyncStart

解説　VSYNC 信号のスタート時点まで待つ。書式はつぎのとおり。

void **VsyncStart**(void)

戻り値　なし

GDC編

GDCの基礎知識

文字表示やグラフィック表示のディスプレイ制御を行っているのがCRTコントローラというハードウェアである。PC-9801のCRTコントローラはGDC（Graphic Display Controller）と呼ばれる。GDCは2個搭載され、1つはテキスト表示制御用で、もう1つはグラフィック表示制御用である。

PC-9801が発表されたときGDCは大きく注目された。なぜならそれまでパーソナルコンピュータに搭載されていたCRTコントローラは表示制御しか行わず、描画はCPUが行っており、描画スピードはけして速いものではなかった。しかし、このGDCは、CRTコントローラ自身に描画機能があり、コマンドとパラメータを出力するだけで、高速に線、円などを描画できたからである。GDCのおかげで、PC-9801のライン描画は高速であり、CADや3次元ゲーム（ライン表示によるゲーム）などに広く利用されることとなった。

しかし、GDCに設定するパラメータは複雑で、実際にプログラムでGDCを操作しようとするとむずかしい。ここでは、GDCの利用方法を説明する前に、GDCの持っている機能の概要とコマンドのフォーマットを紹介する。各コマンドの具体的な利用法に関しては、以降の節を参照してもらいたい。

GDCの描画速度

PC-9801に搭載されているGDCはテキスト制御用、グラフィック制御用ともにμPD7220AというLSIであり、グラフィック制御用のものは動作クロック2.5MHzまたは5.0MHzで動いている。動作クロックが大きいほど描画スピードが速くなり、2.5MHz動作で1ドット描画に要する時間は1.6μ秒となる。これはPC-9801グラフィック画面の左上端から右下端のラインを1秒の間に理論上$1 \div (640 \times 1.6 \times 10^{-6}) =$約1000本描画できるスピードである。ただし、GDCへ出力するコマンドやパラメータは複雑であり、それをCPUが計算し、出力する時間がかなりかかるため、理論上の速さを実現することはできない。

GDCのI/Oポート

グラフィック制御用GDCに関するI/Oポートは、A0HとA2Hの2つのポートのみである。I/OポートA0Hはステータスの読み出しとパラメータの出力に、I/OポートA2Hはデータの読み出しとコマンドの出力に用いられる（Table 1参照）。

Table 1　グラフィック制御用GDCに関するI/Oポート

I/Oポート	Read/Write	意　味
A0H	Read	ステータスレジスタ
	Write	パラメータコードレジスタ
A2H	Read	データレジスタ
	Write	コマンドレジスタ

ステータスレジスタの各ビットの意味はFigure 1のとおりである。

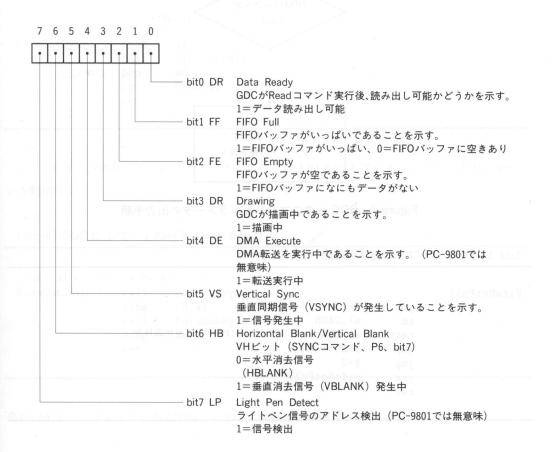



Figure 1　ステータスレジスタ（I/OポートA0H）の各ビットの意味

コマンドの種類によりパラメータの数は異なる。複数のパラメータがある場合は、連続してI/Oポート A0H に出力することによって設定する。

GDC へのコマンドとパラメータの出力

GDC を動作させるためには、多くのパラメータを出力しなければならない。そのため GDC には FIFO バッファと呼ばれるデータバッファ(16個のデータを格納できる)があり、CPU からデータを連続的に出力することができる。したがって GDC にデータを出力するときは、FIFO バッファの格納状況をチェックし、空きがあれば出力することになる。

FIFO バッファの空きを確認する手順を Figure 2 に示す。また、FIFO バッファの空きを確認するプログラムを List 1 に示す。

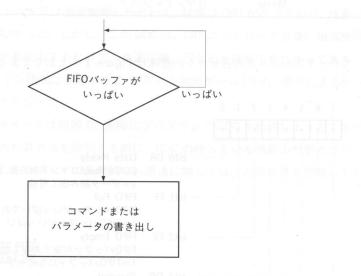



Figure 2 GDC へのコマンドとパラメータの出力手順

List 1 FIFO バッファの空きの確認

```
FifoNotFull:
            jmp     $+2
            in      al,0A0h
            test    al,02h      ; FIFO バッファが満杯か
            jmp     $+2
            jmp     $+2
            jne     FifoNotFull
            ret
```

FIFOバッファの空きを確認したあと、List 2のようにGDCにコマンドを発行する。

List 2 GDCにコマンドを出力

```
;  IN : AL = コマンドデータ
;
CommOut:        push    ax
                call    FifoNotFull
                pop     ax
                out     0A2h,al
                ret
```

コマンドに続けて、そのコマンドに対応したパラメータを出力する。1バイトのパラメータでは、List 3のルーチンを使う。

List 3 GDCに1バイトのパラメータを出力

```
; IN : AL = パラメータデータ
ParaOutByte:    push    ax
                call    FifoNotFull
                pop     ax
                out     0A0h,al
                ret
```

2バイトまたは偶数バイトのパラメータの出力では、List 4の1ワードパラメータ出力のルーチンを利用する。

List 4 GDCに1ワードのパラメータを出力

```
; IN : AX = パラメータデータ
ParaOutWord:    call    ParaOutByte
                xchg    ah,al
                call    ParaOutByte
                xchg    ah,al
                ret
```

奇数バイトのパラメータの出力では、この2つのルーチンを使うことになる。

VRAMとCPU、GDCの関係

GDCはグラフィックVRAMをワード(16ビット)単位で管理する。PC-9801の場合16000ワード×4プレーンのグラフィックVRAMがあることになる。CPUから見たグラフィックVRAMは32000バイト×4プレーンなので、GDCとCPUではVRAMアクセス方法が同じように思えるが(Table 2参照)、ビット配置の違いがあるので注意が必要である。

Table 2　CPUとGDCから見たグラフィックVRAM

グラフィックプレーン	CPU開始アドレス（サイズ）	GDC開始アドレス（サイズ）
プレーン0（青）	A8000H（8000Hバイト）	4000H（4000Hワード）
プレーン1（赤）	B0000H（8000Hバイト）	8000H（4000Hワード）
プレーン2（緑）	B8000H（8000Hバイト）	C000H（4000Hワード）
プレーン3（拡張）	E0000H（8000Hバイト）	0000H（4000Hワード）

たとえば、VRAMのアドレス0000Hと0001Hにつぎのようなデータがあったとする。

VRAMアドレス 0000番地　　0001番地
VRAMデータ　 00001111　　00001010

これらのデータは、ESレジスタにVRAMのセグメントアドレスA800Hが入っているとき、つぎのようにしてCPUからアクセスする。

MOV AX,ES: [0000]

すると、AHレジスタに上位バイト（00001010B）が、ALレジスタに下位バイト（00001111B）が入る。出力についても同様である。

一方、GDCのアドレス0000Hから1ワードを見るとbit0からbit15の並びが、Table 3のようになり、CPUとはビットの並びが逆になるので、GDCを通してデータを読み出すときは注意が必要である。

Table 3　GDCから読み出した1ワードのデータ

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
値	0	0	0	0	1	1	1	1	0	0	0	0	1	0	1	0

GDC のコマンド体系

GDC のコマンド体系はつぎのように 4 つのコマンド群に大別することができる。

- ●動作制御コマンド
  GDC の初期設定や動作モードの設定を行う。
- ●表示制御コマンド
  表示開始／停止、表示開始アドレス、表示領域等の設定を行う。
- ●描画制御コマンド
  GDC の描画に必要なパラメータの設定等を行う。
- ●映像メモリ制御コマンド
  GDC を通して VRAM の読み出し、書き込み等を行う。

動作制御コマンドは、グラフィックの初期設定を行うときに使用するものだが、パラメータがかなり複雑であるためここでは取り上げない。また映像メモリ制御コマンドは、GDC を使ってグラフィック VRAM をアクセスするもので、グラフィック VRAM が CPU から直接アクセスできない機種、または CPU のグラフィック VRAM へのアクセススピードが非常に遅い場合に使われる。PC-9801 は CPU からグラフィック VRAM に容易にアクセスでき、映像メモリ制御コマンドの DMA 関係のコマンドは使用できないためほとんど使われない。ここでは表示制御コマンドと描画制御コマンドについてのみ解説する。

表示制御コマンド

GDC の基本動作は、ディスプレイへの表示制御である。この制御を行うためにつぎのコマンドが用意されている。これらのコマンドにより、グラフィック VRAM のデータを変化させずに表示状態をいろいろ変更できる。

そのうちの 1 つとして、SCROLL コマンドを使った縦方向のスムーススクロールがある。スクロールに関しては、具体的な方法を別の節で解説する。

■START コマンド

グラフィックの表示開始を指示する。コマンドのフォーマットは Figure 3 のとおり。

このコマンドを実行すると、画面が乱れることがある。それを避けるためには、VSYNC 期間中に実行する。また、このコマンドを実行する前に、動作制御コマンド、表示制御コマンドでグラフィックの初期化をしておく必要がある。

Figure 3 START コマンド

■STOP1、STOP2 コマンド

表示停止および外部同期信号の受け付け開始を指示する。コマンドのフォーマットは Figure 4 のとおり。

・コマンド(STOP1)

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	0	0	0CH

・コマンド(STOP2)

7	6	5	4	3	2	1	0	
0	0	0	0	0	1	0	1	05H

Figure 4 STOP1、STOP2 コマンド

■ZOOM コマンド

グラフィック文字描画時の拡大係数を設定する。コマンドとパラメータのフォーマットは Figure 5 のとおり。

それぞれのパラメータの意味は Table 4 のとおりである。

・コマンド

7	6	5	4	3	2	1	0	
0	0	0	0	1	1	0	0	46H

・パラメータ

7	6	5	4	3	2	1	0	
←ZR→				←ZW→				05H

Figure 5 ZOOM コマンド

Table 4　ZOOMのパラメータ

パラメータ	意　味
ZR	表示時の拡大係数。PC-9801では使えない。
ZW	グラフィック文字描画時の拡大係数。

ZR/ZW	倍　数
0000	1倍
0001	2倍
0010	3倍
0011	4倍
1110	15倍
1111	16倍

■SCROLLコマンド

表示開始アドレスおよびグラフィック画面の分割位置を設定する。コマンドのフォーマットとパラメータをFigure 6に示す。

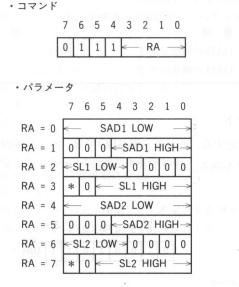



Figure 6　SCROLLコマンド

RAに設定するパラメータを指定し、対応するパラメータをそのあとにGDCに送る。RAはパラメータを出力するごとにGDCによりインクリメントされる。

GDCがグラフィックモードのとき、RAの8～15の値は線種設定などに使われていて使用できないので、画面分割数は2となる(Figure 7参照)。それぞれのパラメータの意味はTable 5のとおりである。

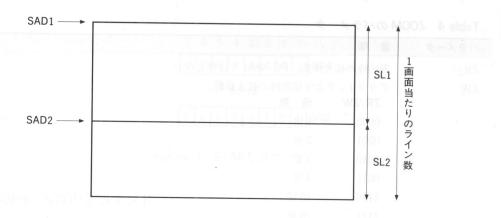



Figure 7　画面分割

Table 5　SCROLL のパラメータ

パラメータ	意　味
SAD	画面の表示開始アドレス
SL	画面の分割表示領域の縦幅を示すライン数 SLの総和≧ L/F（1画面あたりの表示ライン数）
DAD + 2	表示アドレスのインクリメント形態を示す 値　　意　味 0　　DADの増加分が1 1　　DADの増加分が2

■CSRFORM コマンド

1行中のライン数を設定する。テキスト用GDCでは、カーソルの形状を設定する。コマンドに続いて3つのパラメータを送る。コマンドとパラメータのフォーマットはFigure 8のとおり。

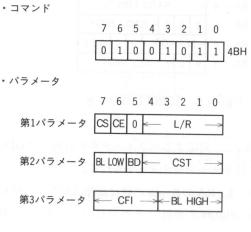



Figure 8　CSRFORM コマンド

それぞれのパラメータの意味はTable 6のとおりである。

Table 6　CSRFORMのパラメータ

パラメータ	意　味
L/R	1行中のライン数 L/R　ライン数 00000　1 00001　2 00010　3 00011　4 ⋮ 11110　31 11111　32 グラフィックモードでは通常 L/R = 0 である。
CE	スレーブ時の外部同期信号受けつけ条件を定義する。 値　意　味 0　外部同期受け付け開始の指示がなされているとき 1　無条件
CS	テキスト表示時のカーソル表示の有無を定義する。 値　意　味 0　カーソル表示なし 1　カーソル表示あり
BD	テキスト表示時のカーソルの点滅の有無を定義する。 値　意　味 0　点滅あり 1　点滅なし
BL	テキスト表示時のカーソルの点滅周期を定義する。 BLの値が1増えると倍の点滅周期になる。 BL = 0 は設定できない。
CST	テキスト表示時のカーソルの表示開始ラインを定義する。
CFI	テキスト表示時のカーソルの表示終了ラインを定義する。 CST/CFI　ライン値 00000　1 00001　2 00010　3 00011　4 ⋮ 11110　31 11111　32

■PITCH コマンド

グラフィック VRAM の横方向のアドレスを定義する。これにより横スクロールが可能となる。コマンドとパラメータのフォーマットは Figure 9 のとおり。

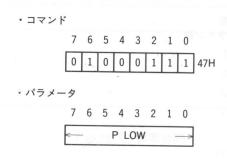



Figure 9 PITCH コマンド

パラメータの意味は Table 7 のとおりである。

Table 7 PITCH のパラメータ

パラメータ	意 味
P LOW	グラフィック VRAM の横幅を定義する。P HIGH は 1 ビットで動作制御コマンドの SYNC コマンドパラメータ 5 で定義する。実際にディスプレイに表示される横幅は動作定義コマンドの SYNC コマンドパラメータ 2 の C/R で定義する。 P　　ライン値 00000000　1 00000001　2 00000010　3 00000011　4 ⋮ 11111110　510 11111111　511

■LPEN コマンド

ライトペン関係のコマンドである。PC-9801 ではほとんど使用しない。

描画制御コマンド

GDC による描画では、つぎの描画制御コマンドを連続で実行してパラメータを設定し、VECTE コマンドで描画開始を指示する。

つぎに、コマンドとパラメータを線、円、四辺形の場合に限定して解説する。

■TEXTWコマンド

直線や円を描画するときの線種データを設定する。コマンドとパラメータのフォーマットをFigure 10に示す。

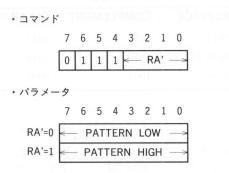



Figure 10　TEXTWコマンド

このパターンデータにより、線、円等の描画が行われる。たとえば、パターンデータがFFFFHで実線となり、AAAAHでは点線となる。

RA'には設定するパラメータを指定する。RA'で指定したパラメータをコマンドの直後に送る。RA'はパラメータを出力するごとにインクリメントされる。

■WRITEコマンド（映像メモリ制御コマンドモードに属する）

ドット修正モードを設定する。コマンドとパラメータをFigure 11、Table 8に示す。

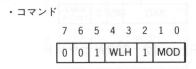



Figure 11　WRITEコマンド

Table 8　WRITEのパラメータ

パラメータ	意味
MOD	ドット修正モード MOD　ドット修正モード 00　REPLACE 01　COMPLEMENT 10　CLEAR 11　SET

それぞれのモードで描画する場合のビットパターンの変化を Table 9 に示す。WLH は、GDC を通してグラフィック VRAM へデータを出力するときの転送形式を示す。

Table 9　ドット修正モードとビットパターンの変化

ドット修正モード	REPLACE	COMPLEMENT	CLEAR	SET
描画前の VRAM パターン	0011	0011	0011	0011
描画パターン	1010	1010	1010	1010
描画結果パターン	1010	1001	0001	1011

■CSRW コマンド

描画実行開始点のVRAMアドレス（GDCにおける）を設定する。コマンドとパラメータのフォーマットは Figure 12 のとおり。

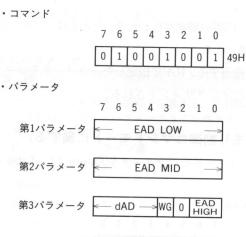



Figure 12　CSRW コマンド

それぞれのパラメータの意味は Table 10 のとおりである。

Table 10　CSRW のパラメータ

パラメータ	意　味
EAD	描画実行ワードアドレス（18 ビットデータ）
dAD	描画ドットアドレス（0～15）
WG	グラフィックモードでは 0

■VECTW コマンド

描画方向、描画種類、描画用各種パラメータを設定する。コマンドとパラメータのフォーマットはFigure 13のとおり。

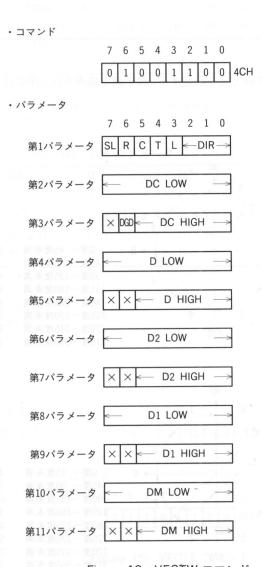



Figure 13　VECTW コマンド

それぞれのパラメータの意味はつぎのとおりである。

第1パラメータのSL、R、C、T、Lには描画種類を設定する。設定値と種類の関係はTable 11のとおり。

Table 11　SL、R、C、T、L パラメータ

	SL	R	C	T	L
直線描画	0	0	0	0	1
円及び弧描画	0	0	1	0	0
四辺形描画	0	1	0	0	0

第1パラメータのDIRには描画方向を設定する。設定値と描画方向の関係はFigure 14のとおり。

・直線の場合

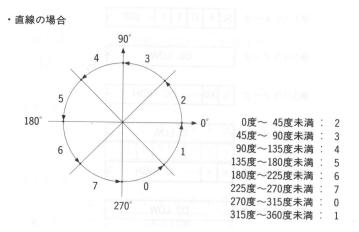
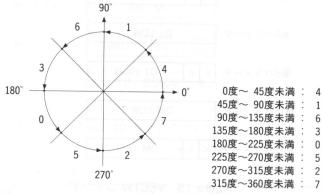
・円の場合

※詳細はNEC GDCユーザーズマニュアル参照

Figure 14　DIRの値と描画方向

第2パラメータのDGDには描画アドレスの進み方を設定する。値の意味はTable 12のとおり。

Table 12　DGDパラメータ

値	意味
0	水平方向以外に描画アドレスが進行する場合、±Pのオフセットアドレスが加わる（GDCクロック2.5MHzのとき）。
1	水平方向以外に描画アドレスが進行する場合、±P/2のオフセットアドレスが加わる（GDCクロック5.0MHzのとき）。
P	PITCHコマンドにより設定するパラメータと同じ。

DC、D、D2、D1、DMは描画パラメータで、描画種類によりTable 13のような意味を持つ。

Table 13　DC、D、D2、D1、DMパラメータ

	DC	D	D2	D1	DM
初期値	0	8	8	−1	−1
直　線	⊿X	2[⊿Y]−[⊿X]	2[⊿Y]−2[⊿X]	2[⊿Y]	×
円	(r/√2)↑	r −1	2(r −1)	−1	0
四辺形	3	A*	B*	−1	A*

⊿X：X座標変位
⊿Y：Y座標変位
A*：第1描画方向変位数
B*：第2描画方向変位数
↑　：切り上げ
[]　：絶対値

■VECTEコマンド

描画の実行を開始する。コマンドのフォーマットはFigure 15のとおり。

・コマンド

7	6	5	4	3	2	1	0	
0	1	1	0	1	1	0	0	6CH

Figure 15　VECTEコマンド

線、円などを初めて描画する際は、すべてのコマンドを実行する必要がある。しかしTEXTWコマンドによる線種データの設定とWRITEコマンドは、1度設定が行われていれば、以後描画する際にコマンド実行を省略することができる。

このGDCによる描画の例を「GDCによる直線の描画」「GDCによる円の描画」「GDCによる四角形の描画」の各節で解説する。

グラフィック画面のスクロール

PC-9801のグラフィックVRAMは、標準で1画面128Kバイトものサイズがある。グラフィックをスクロールさせる場合、このグラフィックVRAM上で転送を行うわけだが、これだけのメモリをCPUで転送するにはかなりの時間が必要となる。CPUを使って転送していては、高速なスクロールを期待することはできない。EGCを利用すれば、かなり高速にグラフィックデータを転送できるが、EGCを搭載していない機種では利用できない。

そこで、ここではGDCのSCROLLコマンドを利用してグラフィック画面の表示開始アドレスを変化させることにより、グラフィックVRAMの内容を変化させずにグラフィック画面をハードウェア的にスクロールさせる方法を解説する。

▶ポイント

- グラフィック画面を2画面に分割し、SCROLLコマンドを用いて、2つの画面を表示する位置を変化させてグラフィック画面のスクロールを実現する。
- SCROLLコマンドの発行はVSYNC信号発生期間中に行い、画面が乱れないようにする。

▶プログラミングテクニック

1 VSYNCを待つ

SCROLLコマンドをグラフィック画面の表示している間に実行すると画面が乱れる。これを避けるため、GDCが画面表示を行っていない期間(VSYNC信号発生期間)を利用してSCROLLコマンドを実行する。そのために、「VSYNC信号の検出」の節で説明した関数を利用してVSYNC発生の開始を待ってから、以降の処理を開始する。

2 スクロールアップかダウンかを判定する

ここで作成するプログラムは、引数により渡されたスクロールライン数の正負を判断し、正ならスクロールアップ、負ならスクロールダウンの処理を行うようにする。

スクロールの処理は、Figure 1に示すように、画面を2つに分割して行う。

標準のグラフィック画面は、Figure 1の（A）のような、画面1のみの状態である。このとき、GDCに設定するSCROLLコマンドのパラメータは、グラフィック画面1の表示開始アドレスSAD1が0、表示ライン数SL1が400、グラフィック画面2の表示ライン数SL2が0となっている。Figure 2にSCROLLコマンドのフォーマットを再掲する。スクロールするときには、この標準状態からFigure 1の（B）のような2画面のグラフィック画面にする。

グラフィック画面1の表示開始アドレスをSAD1＝40（これは1ラインのグラフィックVRAMのワード数である）にし、表示ライン数をSL1＝399にすると、グラフィック画面の2ラインから400ラインの399ラインのグラフィック画面がディスプレイの上端から表示されることになる。そして、ディスプレイの最下ラインには、グラフィックVRAMの1ラインが表示されるように、グラフィック画面2の表示開始アドレスSAD2＝0、表示ライン数SL2＝1にする。これによって、1ラインがスクロールアップされたことになる。

Figure 1　画面の分割

・コマンド

・パラメータ

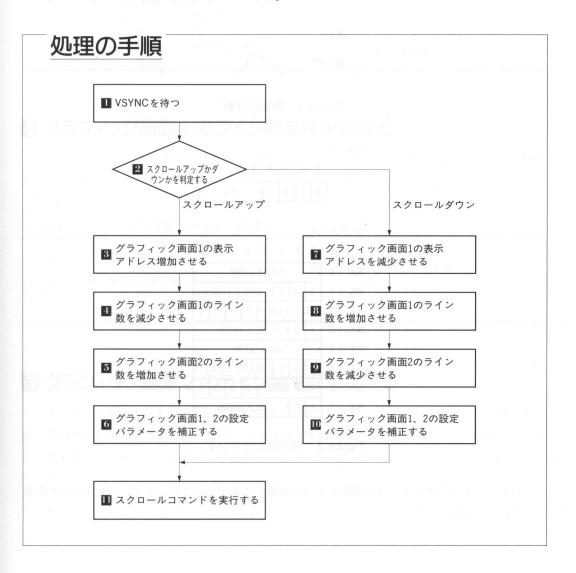
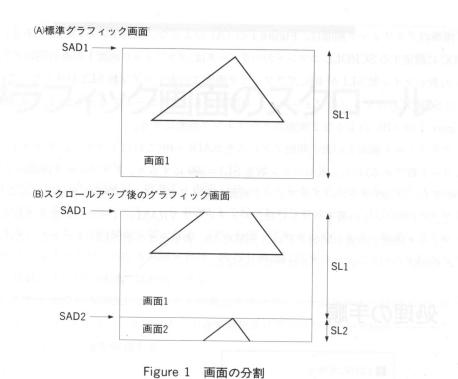
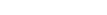



Figure 2　SCROLL コマンド

このように1つのグラフィック画面を2つに分割して表示することにより、グラフィック画面のスクロールを行う。

3 グラフィック画面1の表示アドレス増加させる

このプログラムでは、スクロールアップの場合、グラフィック画面1の表示ライン数を減少させ、グラフィック画面2の表示ライン数を増加させることによりスクロールを行う。スクロールアップして上に消えた画面内容は下端からふたたび表れることになる。

まず、グラフィック画面1の表示開始アドレスをスクロールライン数×40ワードだけ加える。プログラムはList 1のようになる。

List 1　グラフィック画面1の表示アドレス増加

```
sad1l            dw       0                     ; グラフィック画面1の表示開始アドレス
; ax にスクロールするライン数が入っている
                 mov      dx,40
                 mul      dx
                 add      [sad1l],ax
```

4 グラフィック画面1のライン数を減少させる

つぎに、グラフィック画面1の表示ライン数をスクロールライン数分減らす。プログラムはList 2のようになる。

List 2　グラフィック画面1のライン数減少

```
sl1              dw       0                     ; グラフィック画面1のライン数
; ax にスクロールするライン数が入っている
                 sub      [sl1],ax
```

5 グラフィック画面2のライン数を増加させる

グラフィック画面2の表示ライン数をスクロールライン数分加算する。グラフィック画面2の表示開始アドレスは常に0とし、グラフィックVRAMの先頭データがグラフィック画面2の先頭と一致するものとする。プログラムはList 3のようになる。

List 3　グラフィック画面２のライン数増加

```
sl2             dw      0               ; グラフィック画面２のライン数
; ax にスクロールするライン数が入っている
                add     [sl2],ax
```

6 グラフィック画面１、２の設定パラメータを補正する

グラフィック画面１の表示ライン数が減った結果負になった場合は、設定データを補正する。このときグラフィック画面１の表示開始アドレスを400ライン×40ワードだけ引き、グラフィック画面１の表示ライン数に400加え、グラフィック画面２の表示ライン数を400引くことにより、連続したスクロールを実現する。プログラムはList 4のようになる。

List 4　グラフィック画面１、２の設定パラメータの補正

```
        jae     scroll_set              ; グラフィック画面１の表示ライン
                                        ; 数を引いた結果が正または0
        sub     [sl2],400
        sub     [sad11],400*40
        add     [sl1],400
        jmp     SHORT scroll_set
```

以上、スクロールアップのためのパラメータを計算したら、11のスクロールコマンド実行に移る。

7 グラフィック画面１の表示アドレスを減少させる

スクロールダウンの場合、グラフィック画面１の表示ライン数を増加させ、グラフィック画面２の表示ライン数を減少させることによりスクロールを行う。標準グラフィック画面から初めてスクロールダウンするときは、グラフィック画面１の表示ライン数を0、グラフィック画面２の表示ライン数を400としてパラメータを変更する。スクロールダウンして下に消えた画面内容は上端からふたたび表れることになる。

まずは、グラフィック画面１の表示開始アドレスをスクロールライン数×40だけ減らす。プログラムはList 5のようになる。

List 5　グラフィック画面 1 の表示アドレス減少

```
sad1l           dw      0                   ; グラフィック画面 1 の表示開始アドレス
; ax にスクロールするライン数が入っている
                mov     dx,40
                mul     dx
                sub     [sad1l],ax
```

8 グラフィック画面 1 のライン数を増加させる

つぎに、グラフィック画面 1 の表示ライン数をスクロールライン数分加える。プログラムは List 6 のようになる。

List 6　グラフィック画面 1 のライン数増加

```
sl1             dw      0                   ; グラフィック画面 1 のライン数
; ax にスクロールするライン数が入っている
                add     [sl1],ax
```

9 グラフィック画面 2 のライン数を減少させる

グラフィック画面 2 の表示ライン数をスクロールライン数分引く。グラフィック画面 2 の表示開始アドレスは常に 0 とし、グラフィック VRAM の先頭データがグラフィック画面 2 の先頭と一致することになる。プログラムは List 7 のようになる。

List 7　グラフィック画面 2 のライン数減少

```
sl2             dw      0                   ; グラフィック画面 2 のライン数
; ax にスクロールするライン数が入っている
                sub     [sl2],ax
```

10 グラフィック画面 1、2 の設定パラメータを補正する

グラフィック画面2の表示ライン数が、引いた結果、負になった場合には、設定データを補正する。このときグラフィック画面1の表示開始アドレスに400ライン×40ワード加え、グラフィック画面1の表示ライン数から400引き、グラフィック画面2の表示ライン数に400加えることにより、連続したスクロールを実現する。プログラムはList 8のようになる。

List 8　グラフィック画面 1、2 の設定パラメータの補正

```
; ax にスクロールするライン数が入っている
                jae     scroll_set          ; グラフィック画面 1 の表示ライン
                                            ; 数を減算した結果が正または 0
                add     [sl2],400
                add     [sad1l],400*40
                sub     [sl1],400

scroll_set:
```

11 スクロールコマンドを実行する

これで、すべてのパラメータの準備が整ったことになる。用意したパラメータは、グラフィック画面1の表示開始アドレスSAD1と表示ライン数SL1、グラフィック画面2の表示開始アドレスSAD2と表示ライン数SL2である。DAD+2のビットは、GDCのクロックが5MHzのときは1になる（Figure 2参照)。

パラメータを用意したら、GDCに対してSCROLLコマンドを発行する。そのあと、ビット位置を調節しながらパラメータを連続的に出力する。プログラムはList 9のようになる。

List 9　スクロールコマンド実行

```
_vram_offset    dw      0               ; VRAM オフセットアドレス

                mov     ax,[sad1l]
                shl     ax,1
                mov     [_vram_offset],ax

                sub     bx,bx                           ; bx = 0
                mov     cx,4
                mov     si,offset DGROUP:sad1l
                lodsb
                call    SDSsub
                inc     bx
```

```
                lodsb
                call    SDSsub
                inc     bx
                lodsb
                mov     ah,al
                lodsb
                mov     dl,al
                shl     al,cl
                shr     dl,cl
                or      al,ah
                call    SDSsub
                inc     bx
                lodsb
                shl     al,cl
                or      al,dl
                cmp     [_gdc_clock],0
                jz      scf_0
                or      al,40h              ;
scf_0:          call    SDSsub
                inc     bx
                lodsb
                call    SDSsub
                inc     bx
                lodsb
                call    SDSsub
                inc     bx
                lodsb
                mov     ah,al
                lodsb
                mov     dl,al
                shl     al,cl
                shr     dl,cl
                or      al,ah
                call    SDSsub
                inc     bx
                lodsb
                shl     al,cl
                or      al,dl
                cmp     [_gdc_clock],0
                jz      scf_1
                or      al,40h              ;
scf_1:          call    SDSsub
                call    SDSsub
                ret

SDSsub:
                push    ax
                mov     al,70h
                or      al,bl
                out     0A2h,al
                pop     ax
                out     0A0h,al
                ret
```

ワンポイント

■ハードウェアスクロールの問題点

この方法では、グラフィック VRAM の内容は変化しないので、スクロールアップして消えた画面内容は画面下に現れる。したがって、新たな画面を画面下に表示したい場合は、グラフィック VRAM の書き換えが必要となる。また、このスクロールは全画面を対象にしたときのみ可能であり、画面の一部のスクロールには利用できない。

なお、表示アドレスを変更しているため、グラフィック VRAM にデータを書き込むときには、グラフィック VRAM のアドレスに注意しないと、思わぬ位置に書き込んだ内容が表示されることになる。

ライブラリ

GdcScroll

解説 グラフィック画面を GDC の SCROLL コマンドを利用してスクロールする。書式はつぎのとおり。

void **GdcScroll**(int *Line*)

Line スクロールするライン数

Line>=0 のときスクロールアップ

Line<0 のときスクロールダウン

GDC の SCROLL コマンドを利用してグラフィック画面を2分割し、グラフィック画面の表示位置を変更することにより、グラフィック全画面のスクロールを行う。

戻り値 なし

サンプル GDEMO.C

COLUMN

描画方法によるスピードの違い

PC-9801が発売された当初、BASICのLINE文などを使ったベンチマークのテストを行うと、あっというまに描画が終了しPC-9801のスピードに感心したものであった。しかし、この速さはまさにGDCによるハードウェアの描画スピードであり、CPUのスピードではなかったのである。

それでは、この当時のラインの描画速度はどのくらい速かったのだろうか？最近のPC-9801シリーズのPC-98RLで以下のプログラムを実行して時間を測定した。

List A　描画スピードを調べるプログラム

```
void main()
{
        int i,j,k;
        GraphicInit(1,1);
        PaletteInit();
        GraphicCls(1);
        SetPen(0xffff,3);
        for (j=15; j>0 ; j--) {
                for (i=0; i<200 ;i++) {
                        GdcLine(0,0,639,i*2,j);
                        GdcLine(0,399,639,399 - i*2,j);
                        GdcLine(639,0,0,i*2,j);
                        GdcLine(639,399,0,399 - i*2,j);
                }
        }
}
```

PC-9801初代での描画スピードは、ほぼGDC 2.5MHzのフラッシュレス描画相当と考えられる。当時速いと感じていたスピードがかなり遅く、現行機種のグラフィックの描画速度が大幅に向上していることがわかるだろう。

Table A　PC-9801RLによる実行結果

GDCのクロック	描画モード	実行時間
5MHz	4プレーン同時描画	10秒
2.5MHz	4プレーン同時描画	19秒
2.5MHz	1プレーン同時描画	66秒
2.5MHz	1プレーン毎フラッシュレス描画	258秒

GDCによる直線の描画

基礎知識で解説したコマンドを組み合わせると、直線や円、四角形などさまざまな図形が描ける。そのどれもが、パラメータが多少異なるだけで、基本の考え方は同じである。

ここでは、まずもっとも単純な直線の描画を例に、GDCで描画する際の手順を解説する。単純な水平線などでは現在の高速なCPUの場合、直接VRAMにデータを転送して描く方が高速な場合もあるが、斜めの線などの場合はGDCを利用した方が高速に描画できる。

▶ポイント

- 描画制御コマンドをGDCに対し連続して発行し、直線を描画する。
- TEXTWコマンドにより線種、WRITEコマンドによりドット修正モードを設定する。この設定は各直線ごとに行う必要はなく、再び設定するまで変更されない。
- XY座標を正規化することにより、GDCに対して出力するパラメータの値を算出しやすいものとする。
- GDCの描画処理の終了を待ってプログラムを終了する。

▶プログラミングテクニック

1 描画線種を設定する（TEXTWコマンド）

線種を設定する。実線を描く場合は、パターンデータはFFFFHになる（AAAAHで点線）。GDCへのコマンドとパラメータの出力には、基礎知識で解説したCommOut、ParaOutWordルーチンを利用する。プログラムをList 1に示す。

List 1　描画線種の設定（TEXTW コマンド）

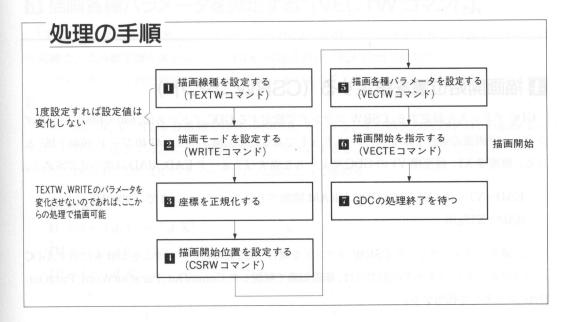
```
mov     al,78h
call    CommOut           ; GDC にコマンドを出力する
mov     ax,0FFFFh
call    ParaOutWord       ; GDC にパラメータを2バイト出力する
```

2 描画モードを設定する（WRITE コマンド）

ドット修正モードを設定する。ドット修正モードは、線種が FFFFH であるため、REPLACE モードでも SET モードでも同様な直線を描くことになる。ここでは、REPLACE モードとして設定することにする。GDC へのコマンド出力には、基礎知識で解説した CommOut ルーチンを利用する。

プログラムを List 2 に示す。

List 2　描画モードの設定（WRITE コマンド）

```
mov     ax,00h            ; REPLACE ドット修正モード
or      al,20h
call    CommOut           ; GDC にコマンドをライトする
```

3 座標を正規化する

直線の開始点(X1, Y1)と終了点(X2, Y2)をX1≦X2であるように正規化する。またX1=X2のときY1>Y2ならY座標を入れ替える。そしてX座標変位⊿X、Y座標変位⊿Yも求めておく。

プログラムをList 3に示す。

List 3　XY座標正規化

```
; IN :   ax = X1, bx = X2, cx = Y1, dx = Y2
         cmp     ax,bx            ; X1 > X2 なら X1 ←→ X2 、Y1 ←→ Y2
         jb      padj_1
         jz      padj_3
         xchg    ax,bx
         xchg    cx,dx
         jmp     SHORT padj_1
padj_3:
         cmp     cx,dx
         jbe     padj_1
         xchg    cx,dx
padj_1:
         mov     [lx01],ax        ; 描画開始点 x 座標
         mov     [lx02],bx        ; 描画終了点 x 座標
         mov     [ly01],cx        ; 描画開始点 y 座標
         mov     [ly02],dx        ; 描画終了点 y 座標
         sub     bx,ax
         mov     [lndx],bx        ; abs(X2 - X1)
         sub     dx,cx
         jae     padj_2
         neg     dx
padj_2:  mov     [lndy],dx        ; abs(Y2 - Y1)
         ret
```

4 描画開始位置を設定する(CSRWコマンド)

GDCアドレスを設定する。CSRWコマンドで設定するGDCアドレスは、640×400ドットのグラフィック画面の場合では1ワード16ドットであるため、横1ラインが40ワード(640÷16)となる。横座標X1、縦座標Y1のGDCアドレスを表すパラメータEAD、dADは次の式で求める。

EAD=Y1×40+X1÷16+GDC VRAM開始アドレス(プレーン0では4000H)
dAD=X1%16

この値をパラメータとしてCSRWコマンドを実行する。実際のプログラムをList 4に示す。GDCへのコマンドとパラメータの出力には、基礎知識で解説したCommOut、ParaOutWord、ParaOut-Byteルーチンを利用する。

List 4　GDC VRAM アドレス設定

```
GdcAdrSet:
        mov     cx,[ly01]           ; cx = 描画開始 y 座標
        mov     ax,40
        mul     cx                  ; EAD = X * 40
        mov     di,ax
        mov     ax,[lx01]           ; ax = bx = 描画開始 x 座標
        mov     dx,ax
        mov     cl,4
        shr     ax,cl
        add     di,ax               ; EAD += X/16
        add     di,4000h            ; EAD += VRAM 開始アドレス
        and     dx,000Fh            ; dAD
        mov     [ead],di
        mov     [dad],dl
;----------CSRW コマンド
        mov     al,49h              ; CSRW コマンド
        call    CommOut
        mov     ax,[ead]
        call    ParaOutWord         ; EAD 出力
        mov     al,[dad]
        mov     cl,4                ; P3 の dAD は bit7～4 に設定する
        shl     al,cl
        call    ParaOutByte         ; dAD 出力
        ret
```

5 描画各種パラメータを設定する（VECTW コマンド）

VECTW コマンドでは多くのパラメータを設定する必要がある。それぞれのパラメータはかなり複雑で、この設定値を求めることがGDC描画において最も面倒な部分であろう。

パラメータは、変数に格納しておき、最後にまとめて出力する。以下、各パラメータについて見ていく。

まず、SL、R、C、T、Lは直線では0、0、0、0、1になる。また描画方向DIRは、Figure 1のフローに従って決定する。これを実際にプログラムにすると、List 5のようになる。

また、DC、D、D2、D1パラメータは次の式によって求める。

DC＝⊿X

D＝2×［⊿Y］－［⊿X］

D2＝2×［⊿Y］－2×［⊿X］

D1＝2×［⊿Y］

DM には何も設定しない。DC 以降のパラメータの設定を List 6 に示す。変数 lndx、lndy には先の正規化により正の値が格納されているものとする。

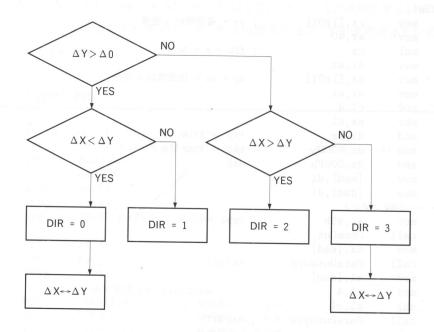



Figure 1 描画方向 DIR の決定

List 5 Direction 設定

```
dirset:
        mov     ax,[ly01]       ; 描画開始点 y 座標
        mov     cx,[ly02]       ; 描画終了点 y 座標
        mov     si,[lndx]       ; △X
        mov     di,[lndy]       ; △Y
        cmp     ax,cx
        jb      dirs_2
        mov     dl,2            ; dir = 2
        cmp     si,di
        ja      dir_set
        inc     dl              ; dir = 3
        jmp     SHORT dirch_set
dirs_2:
        mov     dl,1            ; dir = 1
        cmp     si,di
        jae     dir_set
dirs_3:
        dec     dl              ; dir = 0
        dirch_set:
        mov     [lndx],di       ; △X←→△Y
        mov     [lndy],si
```

```
dir_set:
dir_set: or      dl,08h              ; LINE 描画設定
         mov     [dir],dl            ; dir 設定
         ret
```

VECTWコマンド送出後、以上で求めたパラメータを連続して出力する。ただし、GDCのクロックが5.0MHzのときには第3パラメータのDGDビットを1にする必要がある。実際のプログラムをList 7に示す。

List 6　VECTWパラメータ設定

```
VectwLineSet:
        mov     ax,[lndx]               ; △X
        mov     [dc_reg],ax             ; DC Register 設定
        mov     ax,[lndy]               ; △Y
        shl     ax,1                    ; △Y * 2
        sub     ax,[lndx]
        mov     [d_reg],ax              ; D Register 設定
        mov     ax,[lndy]               ; △Y
        sub     ax,[lndx]               ; △X
        shl     ax,1                    ; 2 * (△Y-△X)
        mov     [d2_reg],ax             ; D2 Register 設定
        mov     ax,[lndy]               ; △Y
        shl     ax,1                    ; △Y * 2
        mov     [d1_reg],ax             ; D1 Register 設定
        mov     [dm_reg],-1             ; DM Register Dumy
        ret
```

List 7　VECTWコマンド送出

```
VectwCommand:
        mov     al,4ch                  ; VECTW コマンド
        call    CommOut
        mov     al,[dir]
        call    ParaOutByte             ; DIR register 設定
        mov     ax,[dc_reg]
        cmp     [_gdc_clock],0          ; GDC クロックが 2.5MHz ならジャンプ
        jz      vec_1
        or      ax,4000h
vec_1:  call    ParaOutWord             ; DC register 設定
        mov     ax,[d_reg]
        call    ParaOutWord             ; D register 設定
        mov     ax,[d2_reg]
        call    ParaOutWord             ; D2 register 設定
        mov     ax,[d1_reg]
        call    ParaOutWord             ; D1 register 設定
vec_2   :ret
```

6 描画開始を指示する（VECTEコマンド）

VECTEコマンド6CHをGDCのI/Oポート6CHに出力して描画を実行させる。プログラムをList 8に示す。

List 8　VECTEコマンド

```
mov     al,6ch
call    CommOut                         ; VECTE 描画開始コマンド
```

7 GDCの処理終了を待つ

GDCが描画中のとき、CPUがグラフィックVRAMにアクセスすると不具合が起こる。また、GRCGを使用してGDC描画を行っているとき、描画途中でGRCGをOFFにすると描画が正常に行われないことになる。そこでGDCの描画状態をチェックして、描画が終了した時点で直線描画ルーチンを抜ける必要がある。ただし、この処理は連続的にGDCを使って描画する際はスピードの点で劣ることになる。

VECTEコマンドが実行されてGDCが描画状態に入ると、**ステータスI/OポートA0Hのbit3**(DR Drawing）がONになる。それと**ステータスI/OポートA0Hのbit1**（FF FIFO Full）の双方をチェックすることにより描画の終了を判断する。

ただし、VECTEコマンドを発行した直後にはI/OポートA0Hのbit3 (DR Drawing)が1にはならないので、コマンドとチェックルーチンの間にウェイトを入れる必要がある。プログラムをList 9に示す。

List 9　GDC終了処理

```
          mov     cx,24              ; 描画が実行されるまでのウェイト
          loop    $
GdcBusy:
          jmp     $+2
          in      al,0A0h            ; GDC ステータスの読み出し
          xor     al,04h
          test    al,0Ch             ; Drawing 状態と FIFO バッファが満杯かのチェック
          jmp     $+2
          jmp     $+2
          jnz     GdcBusy
```

ライブラリ

SetPen

解説　線種（ラインパターン）とドット修正モードを設定する。書式はつぎのとおり。

void **SetPen**(int *Pattern*,int *Writemode*)

Pattern　ラインパターン
Writemode　ドット修正モード

値	モード
0	REPLACE
1	COMPLEMENT
2	CLEAR
3	SET

TEXTW コマンドと WRITE コマンドにより線種とドット修正モードを設定する。この設定を 1 度行えば GDC による描画ではすべてこの設定値により描画が行われる。したがってこの設定を変更するときのみこの関数を実行するものとする。モードの意味は基礎知識参照。

戻り値　なし

サンプル　GDEMO.C

GdcLine

解説　グラフィック画面に GDC を利用して線を描画する。書式はつぎのとおり。

void **GdcLine**(int *X1*,int *Y1*,int *X2*,int *Y2*,int *Color*)

X1, Y1　開始点座標
X2, Y2　終点座標
Color　色番号

GDC のクロックに対応したパラメータを用意し、描画制御コマンドを実行することにより直線の描画を行う。EGC 搭載の PC-9801 では 4 プレーン同時書き込みによる高速描画が可能である。なお、GdcLine は、線種の変更はできないので、線種を設定するときは、SetPen を呼ぶ。

戻り値　なし

サンプル　GDEMO.C

GDCによる円の描画

直線に続き、GDCの描画制御コマンドを使ってグラフィック画面に高速に円を描画する方法を解説する。基本的な手順は直線の描画と同様であるが、GDCのコマンドでは1度に1/8の円弧しか描画できないので、描画する方向（DIRパラメータ）を変化させて円を描画する。

なお、1ピクセルの縦横比が等しく形が正方形のときのみ正円としてディスプレイに表示される。200ラインなど縦横比が異なるときは楕円となる。

▶ポイント

- 描画制御コマンドをGDCに対し連続して発行し、円を描画する。
- 線種とドット修正モードの設定は、直線描画と共通に利用する。
- 1度に描画できるのは1/8の円弧であるため、描画開始位置のXY座標を求める処理からVECTEコマンドまでの処理を8回実行することにより円の描画を行う。
- 最後の1/8の円弧の描画終了を待って処理を終了する。

▶プログラミングテクニック

1 描画線種を設定する（TEXTWコマンド）

線種を設定する。設定は直線描画の場合と同様に行う。

2 描画モードを設定する（WRITEコマンド）

ドット修正モードを設定する。設定は直線描画の場合と同様に行う。

ただし、COMPLEMENTモードにすると、1/8の円孤が隣の1/8の円弧と重なる部分で1ドットの点が抜けてしまうので、避けるべきである。

3 描画開始座標を決定する

GDCが1度に描ける円は全周の1/8の円弧でしかない。そこで8回の描画処理により円を描く(Figure 1参照)。

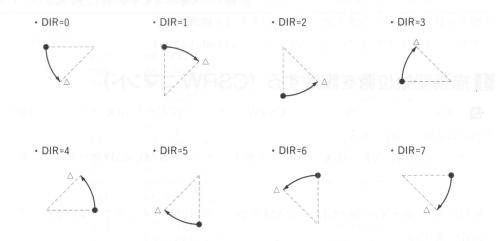
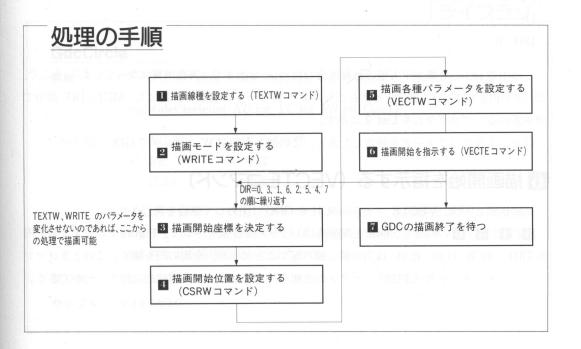



Figure 1　GDC が 1 度に描ける円弧

List 1　DIR = 0,3 の円弧描画におけるスタート座標の決定

```
mov     ax,[xx]         ; 中心 X 座標
sub     ax,[rr]         ; 半径を引く
mov     [lx01],ax       ; スタート X 座標設定
mov     ax,[yy]         ; スタート Y 座標は中心 Y 座標と同じ
mov     [ly01],ax
```

スタート座標は、円の中心から上下左右に半径分だけ離れた4つの地点である。1つのスタート座標を計算したら、その値を使って2つの1/8円を描画する。

スタート座標を求めるプログラムを前ページのList 1に示す。

4 描画開始位置を設定する（CSRWコマンド）

3 で求めたスタート座標によって、CSRWコマンドで設定するGDCアドレスを計算する。計算方法は直線と同様である。

スタート座標X1、Y1のGDCアドレスを表すパラメータEAD、dADはつぎの式によって求める。

EAD＝Y1×40＋X1÷16＋GDC VRAM開始アドレス（プレーン0では4000H）

dAD＝X1％16

この値をパラメータとしてCSRWコマンドを実行する。

5 描画各種パラメータを設定する（VECTWコマンド）

円描画におけるVECTWコマンドのパラメータは、つぎの式で求められる。

SL、R、C、T、L＝0、0、1、0、0

DC＝半径÷$\sqrt{2}$（小数点以下は切り上げる）

D＝半径－1

D2＝2×（半径－1）

D1＝－1

DM＝0

この中でDCの計算が平方根の計算を含むため、そのままでは実数演算になってしまう。そこで、これを(半径×14142)÷20000とすることにより16ビットの整数の計算にし、MUL、DIV命令で値を求める。プログラムをList 2に示す。

以上のようなパラメータを設定したあと、この値をVECTWコマンドでGDCに出力する。

6 描画開始を指示する（VECTEコマンド）

直線描画と同様、VECTEコマンド6CHをGDCに出力して描画を実行させる。

3、4、5、6の処理を、DIRと開始点のパラメータを変化させて、Figure 1に示す8個の円弧DIR＝(0,3)、(1,6)、(2,5)、(4,7)の順に繰り返すことにより、全周の円を描く。このときコマンド、パラメータの出力はFIFOバッファの状態だけをチェックして連続的に行うことができる。

List 2　円描画 VECTW コマンドパラメータ設定

```
VectwCircleSet:
        mov     ax,[rr]              ; 半径
        mov     dx,14142
        mul     dx
        mov     cx,20000             ; 半径 / sqrt(2)
        div     cx
        inc     ax
        mov     [dc_reg],ax          ; DC Register 設定
        mov     ax,RR
        dec     ax
        mov     [d_reg],ax           ; D Register 設定
        shl     ax,1
        mov     [d2_reg],ax          ; D2 Register 設定
        mov     [d1_reg],-1          ; D1 Register 設定
        mov     [dm_reg],0           ; DM Register 設定
        ret
```

7 GDC の描画終了を待つ

8 個の円弧の描画コマンドを発行したあと、描画中かどうかのチェックを行い、すべての円弧描画の終了を確認する。チェックの方法は直線描画の場合と同様である。

ライブラリ

GdcCircle

解説　グラフィック画面に GDC を利用して円を描画する。書式はつぎのとおり。

void **GdcCircle**(int *XC*,int *YC*,int *RR*,int *Color*)

XC, *YC*	中心点座標
RR	半径
Color	色番号

GDC のクロックに対応したパラメータを用意し、描画制御コマンドを実行することにより円を描画する。EGC 搭載の PC-9801 では 4 プレーン同時書き込みによる高速描画が可能である。

戻り値　なし

サンプル　GDEMO.C

GDCによる四角形の描画

直線、円に続き、GDCの描画制御コマンドを使ってグラフィック画面に四角形を描画する方法を解説する。四角形の描画コマンドでは、1度に水平線と垂直線の4本を引くので、直線のコマンドで描画するよりさらに高速になる。ただし、X座標やY座標が同じになった場合、四角形の描画コマンドでは直線は描けないので、直線の描画として扱わなければならないなど、いくつかの注意が必要になる。

▶ポイント

- 描画制御コマンドをGDCに対し連続して発行し、四角形を描画する。
- 1度の描画で水平線2本、垂直線2本を描く。
- 線種とドット修正モードの設定は、直線描画と共通に利用する。
- 上左端のXY座標と右下端のXY座標を求める。
- 描画終了を待って処理を終了する。

▶プログラミングテクニック

1 描画線種を設定する（TEXTWコマンド）

線種を設定する。設定は直線描画の場合と同様に行う。

2 描画モードを設定する（WRITEコマンド）

ドット修正モードを設定する。設定は直線描画の場合と同様に行う。

3 座標を正規化する

まず、特殊な条件を判別する。左上座標を(X1, Y1)、右下座標を(X2, Y2)としたとき、X1=X2のときは垂直の直線、Y1=Y2のときは水平の直線であるので直線描画ルーチンへジャンプする。四角形描画コマンドでは、直線の描画はできない。そのあとに、X1<X2かつY1<Y2であることを確認し、そうでないときは値を入れ替える。プログラムリストをList 1に示す。

List 1　XY 座標の決定

```
          mov     ax,[lx01]        ; 左上 x 座標 X1
          cmp     ax,[lx02]        ; 右下 x 座標 X2
          jz      line_entry       ; X1 = X2 なら直線描画ルーチンへ
          mov     ax,[ly01]        ; 左上 y 座標 Y1
          cmp     ax,[ly02]        ; 右下 y 座標 Y2
          jz      line_entry       ; Y1 = Y2 なら直線描画ルーチンへ
          mov     ax,[lx01]
          mov     dx,[lx02]
          cmp     ax,dx            ; X1 > X2 なら入れ替え
          jbe     gbox_1
          xchg    ax,dx
          mov     [lx01],ax
          mov     [lx02],dx
gbox_1:
          sub     dx,ax
          mov     [lndx],dx        ; △X
          mov     ax,[ly01]
          mov     dx,[ly02]
          cmp     ax,dx            ; Y1 > Y2 なら入れ替え
          jbe     gbox_2
          xchg    ax,dx
          mov     [ly01],ax
          mov     [ly02],dx
gbox_2:
          sub     dx,ax
          mov     [lndy],dx        ; △Y
```

4 描画開始位置を設定する（CSRWコマンド）

左上座標（X1，Y1）のGDCアドレスを計算し、CSRWコマンドでGDCに設定する。計算方法は直線と同様である。

5 描画各種パラメータを設定する（VECTWコマンド）

四角形描画におけるVECTWコマンドのパラメータは、つぎのようにして求める。

SL、R、C、T、L＝0、1、0、0、0
DC ＝3
D ＝縦の長さ（ドット単位）
D2 ＝横の長さ（ドット単位）
D1 ＝−1
DM ＝横の長さ（ドット単位）
DIR＝0

プログラムをList 2に示す。

List 2　VECTWコマンドパラメータ設定

```
VectwBoxSet:
        mov     [dir],40h           ; Box Command 設定
        mov     [dc_reg],3          ; DC Register 設定
        mov     ax,[lndy]
        mov     [d_reg],ax          ; D Register 設定
        mov     [dm_reg],ax         ; DM Register 設定
        mov     ax,[lndx]
        mov     [d2_reg],ax         ; D2 Register 設定
        mov     [d1_reg],-1         ; D1 Register 設定
        ret
```

以上のようなパラメータを設定したあと、この値をVECTWコマンドでGDCに出力する。

6 描画開始を指示する（VECTEコマンド）

直線描画と同様、VECTEコマンド6CHをGDCに出力して描画を実行させる。

7 GDCの処理終了を待つ

描画コマンドを発行したあと、描画中かどうかのチェックを行い、すべての描画の終了を確認する。チェックの方法は直線や円の場合と同様である。

▶ワンポイント

■GDCとCPUの四角形描画

四角形の描画は、CPUによる描画でもかなり速く描くことができるので、最近のCPU速度の高いマシンではCPU描画の方がGDC描画より速くなる可能性がある。

ただし、GDCによる描画では水平に置かれた四角形の他に、45度傾いた四角形の描画が可能である。あまり実用には使われない形状であるが、この描画ではGDC描画の方がはるかに速い。

ライブラリ

GdcBox

解説　グラフィック画面にGDCを利用して四角形を描画する。書式はつぎのとおり。

void **GdcBox**(int *X1*,int *Y1*,int *X2*,int *Y2*,int *Color*)

X1,Y1　開始点座標
X2,Y2　終点座標
Color　色番号

GDCのクロックに対応したパラメータを用意し、描画制御コマンドを実行して四角形を描画する。EGC搭載のPC-9801では4プレーン同時書き込みによる高速描画が可能である。

戻り値　なし
サンプル　GDEMO.C

GDCによる4プレーン描画

いままで解説したGDC描画コマンドにおける描画はすべて1プレーンに対してのものである。しかし今までの説明では、プレーン0(青プレーン)のみにしか描かれない。しかし、PC-9801のグラフィック画面は4プレーン(旧機種は3プレーン)あり、実際のプログラムではGDCの描画をそれぞれのプレーンに対して行う必要がある。

EGCタイプのPC-9801では、このように4プレーン各々への書き込みを、GRCG互換モードを使うことでGDCにより4プレーン同時描画が行える。ここでは、GDCだけで4プレーンに順次書き込む方法と、EGCのGRCG互換モードを使って4プレーン同時に書き込む方法をそれぞれ解説する。なお、GRCG互換モードに関しては「EGCの基礎知識」を参照してほしい。

▶ポイント

- EGCを搭載していない機種(初期タイプ、VMタイプ)では、1プレーンごとにWRITEコマンドによりドット修正モードを変更し、指定の色番号の描画を行う。
- EGCを搭載した機種(EGCタイプ)では、GRCG互換モードを利用してGDCによる4プレーン同時描画を行う。

▶プログラミングテクニック

1 GDCへのパラメータを計算する

4プレーンに順に描画する場合でも、同時に描画する場合でも、GDCコマンドに与えるパラメータをあらかじめ計算しておく。この具体的な方法は、直線、円、矩形に関してこれまでの節で解説してきた。具体的な方法は各々の節をみてほしい。

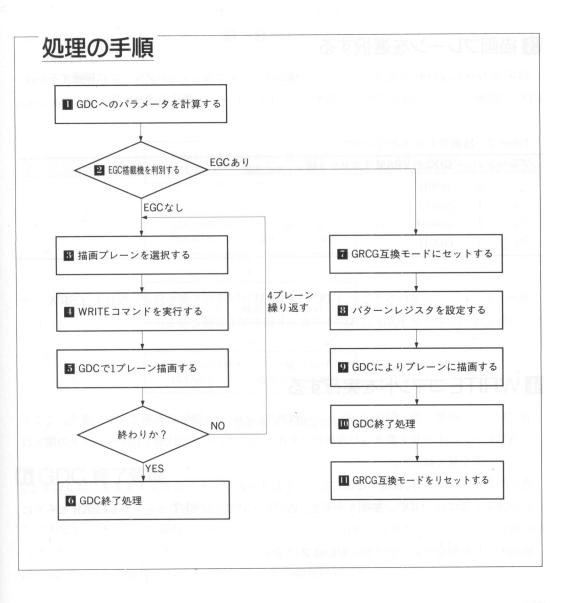
2 EGC搭載機を判別する

ポイントで説明したように、EGCが搭載された機種では、GDCからGRCG互換モードを使って4プレーン同時に描画できる。そこで、まずEGCを搭載しているかどうかを判別し、搭載していればGRCG互換モードを使って4プレーン同時描画を行い、搭載していなければ4プレーンに

GDCで順次描画を行うことにする。

EGCの有無は、Table 1のシステム共通域0000:054DHのbit6を参照して判別する。

Table 1　EGCが利用できるか判別するシステム共通域

アドレス	意　味
0000:054DH	グラフィックモード（PRXDUPD） bit6　EGC使用可否 1　可 0　不可 （NS/Eではレジューム機能を使うとEGCが使えなくなる）

このビットをList 1のようにして判定し、EGCが存在すればGRCG互換モードで4プレーン同時描画を行う。

List 1　EGCを利用できるかの判定

```
            mov     ax,0
            mov     es,ax
            test    BYTE PTR es:[054Dh],040h
            jz      not_egc                     ; EGC 無し
exist_egc:                                      ; EGC 有り
```

3 描画プレーンを選択する

EGCが存在しない場合は、1プレーンづつ描画することになる。どのプレーンに描画するかは、GDCの描画アドレスの設定で行う。描画アドレスとプレーンの関係をTable 2に示した。

Table 2　描画アドレスとプレーン

プレーン	GDCのVRAMオフセット値
プレーン0	4000H
プレーン1	8000H
プレーン2	C000H
プレーン3	0000H

描画するプレーンが変わるごとに、GDCのVRAMオフセット値を設定しなおす。CSRWコマンドに与えるパラメータEADはこのオフセット値を加えた値となる。

4 WRITEコマンドを実行する

各プレーンの状態と色の関係はプレーンの枚数や、パレット機能の有無で大きく違う。ここでは、各プレーンに1か0を書き込む方法だけを述べる。プレーンの状態と色やパレットの関係はパレットの節を見て欲しい。

色を設定するためには、設定する色に応じてそれぞれのプレーンのドットを1または0にする。これを設定するには、GDCの描画モードを、各プレーンごとにSETモードやCLEARモードに切り替える。

描画モードを設定するプログラムをList 2に示す。

List 2　描画モードの切り変え

```
; dx に色コードが入っている
        mov     ax,dx
        and     ax,1
        or      al,022h
        call    CommOut                 ; Write  mode コマンド
        shr     dx,1
```

5 GDC で1プレーン描画する

描画するプレーンとモードが設定されたら描画する。GDC を使うので一般的には VECTW コマンドを利用することになる。そして、3 と 4 を 4 プレーン繰り返すことによって 4 プレーンすべてに描画する。

以上の手順により、GDC を使って 4 プレーンに描画する。Figure 1 に、色番号 5（シアン）の描画を行う場合の例を示す。

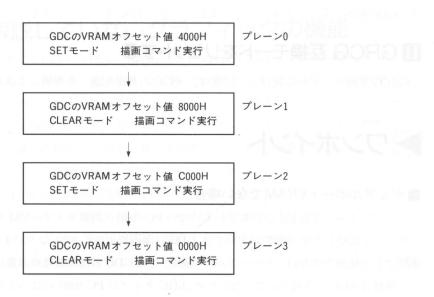



Figure 1　色番号 5 の場合の描画手順

6 GDC 終了処理

1プレーン描画するごとに描画終了をチェックすると、無駄が多く、スピードが遅くなる。そこで、それぞれのプレーンの描画では終了処理をスキップし、GDC の FIFO の状態をチェックするのみでコマンドとパラメータを出力する。そしてプレーン 3 の描画コマンドが発行された後、すなわち全プレーンの描画コマンドが発行された後に描画終了処理を行う。

7 GRCG 互換モードにセットする

EGC が搭載された PC-9801 では EGC の GRCG 互換モードを使うことによって、CPU による 4 プレーン同時描画が GDC の描画でも利用できる。描画が 1 プレーンだけで済むため、高速に描画できる。詳しくは「EGC の基礎知識」を参照のこと。

8 パターンレジスタを設定する

描画に先だって、パターンレジスタを設定する。設定方法は「GRCG の基礎知識」の節を参考にする。

9 GDC によりプレーンに描画する

設定が済んだら実際に描画する。別節で示した GDC による描画を参考に必要な描画を行う。

10 GDC 終了処理

GDC の描画中に EGC が OFF にされると、途中から色違いのドットが描かれてしまう。そのため、GDC 描画が終了したかどうかを確認してから EGC をリセットする。

11 GRCG 互換モードをリセットする

GRCG 互換モードから抜ける。詳細は、「EGC の基礎知識」を参照して欲しい。

▶ワンポイント

■デュアルポートVRAMでない場合

デュアルポート VRAM が搭載されていない PC-9801（初期タイプ〜VM タイプのうちデュアルポート VRAM でない機種）においては、GDC は画面表示を乱さないようにするためには VSYNC 期間にしか描画できない（フラッシュレス描画）ため GDC の描画速度が非常に遅くなる。そのような機種は EGC も搭載されていないため、EGC タイプの PC-9801 に比べて描画速度が圧倒的に遅いことになる。同じ GDC のクロックで描画してもそのマシンの違いで約 12 倍以上の描画速度の差がある。

■ライブラリ関数と同時描画について

ここではとくに例示しなかったが、いままでの GDC 描画のライブラリ関数はすべてこの 4 画面同時描画を行うようになっている。

COLUMN

解説していないグラフィックの機能

4プレーン描画でも利用したEGC（Enhanced Graphic Chager）の内部情報については、『PC-98XL/XAテクニカルデータブック』と『PC-9800テクニカルデータブック増補版』（共にアスキー出版局刊）にわずかに解説がある他はまったく公開されていない。したがって、本書におけるEGCの解説は、数多くのテスト用プログラムを作成し、その結果を考察することにより、EGCの動作を解明した結果に基づき説明している。互換機の問題などあるのかもしれないが、是非正式な内部情報を公開してほしいものだ。

ところで今回は、紙面と時間の問題からEGCの説明はまだそのさわりの部分だけで終わっている。実用的なグラフィック画面のブロック転送は、添付ディスク中のプログラムソースのみの解説となってしまった。また、EGCの機能により高速な描画が実現できるグラフィックキャラクタの重ねあわせや、背景上のスムースな移動など、ほかにも解説したい内容がまだまだ数多くある。

さらに、PC-H98シリーズに搭載されたAGDC（Advanced Graphic Display Controller）は、複雑なパラメータなしでペイントからポリゴン描画まで行える優れたハードウェアである。また、こういったデバイスのほかにも、グラフィック描画関連のアルゴリズムなど、グラフィックに関しては本書だけでは説明不足の点がある。

現在、こうしたグラフィック関連のテクニックの広く詳しい解説書をみなさまのもとへ届ける準備を着々と進めているところである。もう少しお待ちいただきたい。

グラフィック画面への文字表示

PC-9801ではANKキャラクタROMと漢字ROMに文字のパターンデータが納められており、ハードウェアがそのデータを使って、テキストVRAM上のデータに対応する文字を画面に表示するようになっている。これら、文字のデータのはいったROMをキャラクタジェネレータという。

キャラクタジェネレータに納められた文字パターンデータをCPUが読み出してグラフィックVRAMに書き込むことにより、ドット単位の位置への表示、4096色中16色表示など、自由度が高く表現力豊かな文字表示ができる。この節ではそれらのテクニックを使ったグラフィック画面への文字表示について解説する。

▶ポイント

- 文字パターンデータを読み出すには、I/Oポートを利用する方法と、CGウィンドウを利用する方法の2つがある。CGウィンドウを利用するほうが高速だが、EGCタイプのPC-9801でしか使えないため、事前にEGCの存在をチェックし、EGCがないときにはI/Oポートを利用する。
- I/Oポートからは1バイトずつしか読み出せないため、ビットマップの位置データを変化させながら繰り返しパターンデータを読む。
- CGウィンドウでは、I/Oポートのアクセスと違い32バイトのパターンデータがメモリ上に配置されるため、連続的なメモリの読み出しでパターンデータを読む。
- 読み出したパターンデータを高速にグラフィック画面に描画するには、GRCG(グラフィックチャージャ)を使用する。したがって、ここで示す関数は初期型タイプのPC-9801では動作しない。
- 横位置が8ドット単位ではない場合は、パターンデータ自体をシフトして変更して描画する。

▶ プログラミングテクニック

1 CG ウィンドウの有無を調査する

キャラクタジェネレータに格納されたパターンデータを読み出すには2つの方法がある。

1つはI/Oポートから直接読み込む方法である。この方法は、ノーマルモードのPC-9801で使用できるが、I/Oポートから1バイトづつ読み込むため速度が遅い。

もう1つはCGウィンドウを利用する方法である。これはメモリアクセスでデータを読み込めるため高速であるが、CGウィンドウはEGC搭載の機種にしか備わっておらず、汎用性の面で不利である。

CGウィンドウを利用したほうが高速なため、ここでは、EGCが存在すればCGウィンドウを、存在しなければI/Oポートを利用することにする。EGCの有無は、Table 1に示すように、**システム共通域の0000:054Dのbit6を参照して判別する。**

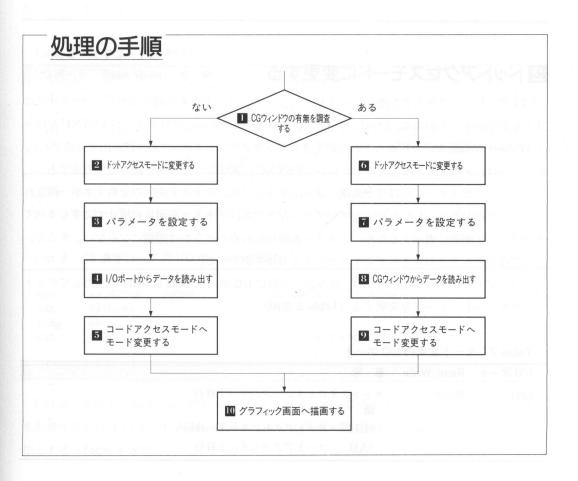



Table 1　EGC の有無

アドレス	意　味	
0000:054DH	グラフィックモード（PRXDUPD）	
	bit 6	EGC 使用可否
	1	可
	0	不可

この判別プログラムは List 1 のようになる。

List 1　EGC の存在調査

```
          sub     ax,ax                       ; ax = 0
          mov     es,ax
          test    BYTE PTR es:[054Dh],01000000b
          jz      io_read                     ; I/O ポートからの読み出しルーチンへ
cg_read:                                      ; CG ウィンドウからの読み出しルーチン
```

2 ドットアクセスモードに変更する

I/O ポートにアクセスする場合でも、CG ウィンドウにアクセスする場合でも、アクセスするには 2 とおりのモードがある。1 つは、コードアクセスモードといい、プログラムが VSYNC を待って PC-9801 が画面表示を行なっていないときにキャラクタジェネレータから読み出す方法である。もう一つは、ドットアクセスモードといい、VSYNC に関係なくデータを読み出す方法である。

ドットアクセスモードのほうが高速に読み出せるのだが、テキスト画面の全角文字が一瞬乱れるという欠点がある。しかしゲームプログラムなどでは、テキスト画面を利用せず文字もすべてグラフィック画面に表示するため、テキスト画面が乱れるかどうかは問題にならない。そこで、ここではドットアクセスモードでアクセスする方法を紹介する。

ドットアクセスモードを使うには、読み出しの前に **I/O ポート 68H** に 0BH を出力してドットアクセスモードへモードを変更する（Table 2 参照）。

Table 2　モード変更の I/O ポート

I/O ポート	Read/Write	意　味	
68H	Write	キャラクタジェネレータのモード移行	
		値	モード
		0BH	ドットアクセスモードへ移行
		0AH	コードアクセスモード移行

モード変更のプログラムはList 2のようになる。

List 2　ドットアクセスモードへの変更

```
mov     al,0Bh              ; ドットアクセスモード
out     68h,al              ; アクセスモードの変更
```

3 パラメータを設定する

どの漢字のパターンを得るかは、I/OポートA1HにJISの漢字コードの下位バイトを、A3Hに上位バイトから20H引いた値（テキストVRAMへ書き込む値と同じ）をそれぞれ出力して指定する(Table 3参照)。また、1バイトANK文字の文字パターンを得るには、ASCIIコードをI/OポートA3Hに出力し、I/OポートA1Hに0を出力する。ただし、ANK文字はドットアクセスモードではVSYNC期間以外は読み出せないため、コードアクセスモードでVSYNCを待って読み出す必要がある。

Table 3　文字コード設定のI/Oポート

I/Oポート	Read/Write	意　味
A1H	Write	文字コードのキャラクタジェネレータへの設定。全角文字の場合JIS漢字コードの下位バイトを書き、ANK文字の場合は0を書く。
A3H	Write	文字コードのキャラクタジェネレータへの設定。全角文字の場合JIS漢字コードの上位バイト－20Hを書き、ANK文字の場合はキャラクタコードを書く。

例として、List 3にJISコード4770H‘柏’の漢字コードを設定するプログラムを示す。

List 3　JIS漢字コードのI/Oポートへの出力

```
mov     ax,4770H - 2000H        ; '柏' JIS漢字コード 4770H
out     0A1h,al                 ; 下位バイト出力 70H
xchg    ah,al
out     0A3h,al                 ; 上位バイト出力 47H - 20H
```

I/Oポートによる方法では、文字パターンを1バイトずつ取得するが、そのパターンのどの部分を取得するかは、I/OポートA5Hに文字の上からのライン数と左右の情報を出力することにより指定する（Table 4参照）。

Table 4　位置指定のI/Oポート

I/Oポート	Read/Write	意　味	
A5H	Write	パターンデータ位置の設定	
		ビット	意　味
		bit 0～bit 4	文字パターンデータの上から何ライン目をリードするのか指定。ただし、bit4は常に0にする
		bit 5	全角漢字の左（1）右（0）の指定
		bit 6～bit7	常に0

1バイトANK文字の文字パターンを得るには、ライン数は必要だが、左右のデータは必要がない（実際には、左右どちらを指定しても無意味）。

たとえば、漢字の最上段左の1バイトのデータを読み出す場合、位置データはList 4のようにして設定する。

List 4　位置データの設定

```
mov     al,20H                    ; LINE 0 左 データ
out     0A5h,al
```

4 I/Oポートからデータを読み出す

以上のデータを出力したあとでI/OポートA9Hを読み出すと、文字のパターンのうち1バイトを取得することができる（Table 5参照）。

Table 5　文字パターンデータのI/Oポート

I/Oポート	Read/Write	意　味
A9H	Read	文字パターンデータが現れる
	Write	文字パターンデータをユーザー定義文字領域に登録する

位置データを変化させながら続けて読めば、文字全体のパターンデータが得られる。

Figure 1に、全角文字'柏'とANK文字'A'の文字データの例を示す。

●全角文字のパターンデータ
例：'柏'JIS漢字コード 4770H

	左のデータ								右のデータ								
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	:0810H
1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	:0810H
2	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	:0820H
3	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	:3EFFH
4	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	:0881H
5	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	:1881H
6	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	1	:1C81H
7	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	1	:2A81H
8	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	1	:2881H
9	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	:28FFH
10	0	0	1	0	1	0	0	0	1	1	1	1	1	1	1	1	:4881H
11	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	1	:0881H
12	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	:0881H
13	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	:0881H
14	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	:0881H
15	0	0	0	0	1	0	0	0	1	1	1	1	1	1	1	1	:08FFH

●ANK文字のパターンデータ
例：'A'ANKコード 41H

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	0	0	0	0	0	0	0	0	:00H
1	0	0	0	0	0	0	0	0	:00H
2	0	0	0	0	1	0	0	0	:08H
3	0	0	0	1	0	1	0	0	:14H
4	0	0	1	0	0	0	1	0	:22H
5	0	1	0	0	0	0	0	1	:41H
6	0	1	0	0	0	0	0	1	:41H
7	0	1	0	0	0	0	0	1	:41H
8	0	1	0	0	0	0	0	1	:41H
9	0	1	1	1	1	1	1	1	:7FH
10	0	1	1	0	0	0	0	1	:41H
11	0	1	0	0	0	0	0	1	:41H
12	0	1	0	0	0	0	0	1	:41H
13	0	0	0	0	0	0	0	0	:00H
14	0	0	0	0	0	0	0	0	:00H
15	0	0	0	0	0	0	0	0	:00H

Figure 1　読み出したキャラクタコードデータ

たとえば、全角漢字の'柏'（JIS漢字コード4770H）の最上段左のデータを得るには、I/Oポート A1Hに漢字下位バイトデータ70Hを、I/OポートA3Hに漢字上位バイト47H－20Hを出力し、I/OポートA5Hに最上段左の位置データ20Hを出力する。そのあとでI/OポートA9Hを読むと、パターンデータ08Hが取得できる（List 5参照）。

List 5 ‘柏’の最上段左のパターンデータの読み出し

```
mov     ax,4770H - 2000H          ; ’柏’ JIS 漢字コード 4770H
out     0A1h,al                   ; 下位バイト出力 70H
xchg    ah,al
out     0A3h,al                   ; 上位バイト出力 47H - 20H
mov     al,20H                    ; LINE 0 左 データ
out     0A5h,al
in      al,0A9h                   ; パターンデータの読み出し 08H
```

続けて、I/O ポート A5H につぎの位置データを出力して、I/O ポート A9H からパターンデータを読み取っていく。

5 コードアクセスモードに変更する

パターン読み出し処理が終了した時点で、I/O ポート 68H に 0AH を出力してコードアクセスモードに戻し、漢字テキストが表示できるモードにする（Table 2 参照）。リストを List 6 に示す。

List 6 コードアクセスモードに変更する

```
mov     al,0Ah            ; コードアクセスモード
out     68h,al            ; アクセスモードの変更
```

以上で、I/O ポートからのデータの読み出しは完了した。このあとは、グラフィック画面への描画に移る。

6 ドットアクセスモードに変更する

EGC を搭載した PC-9801 には、CG ウィンドウと呼ばれる、キャラクタのパターンイメージをある特定のアドレスに出現させれられるような機構が備っている。CG ウィンドウはメモリ上に出現するため、連続してパターンを転送することができ、I/O ポートを利用するより高速に読み出せる。

CG ウィンドウへのアクセスにドットアクセスモードを使う場合には、I/O ポートから読み出すときと同様に、読み出す前に I/O ポート 68H に 0BH を出力してドットアクセスモードへ移行する（Table 2、List 2 参照）。

7 パラメータを設定する

CGウィンドウにパターンデータを出現させるには、I/Oポートを利用したときと同様に、Table 2に示したI/OポートA1HにJIS漢字コードの下位バイトを、I/OポートA3Hに上位バイトから20H引いた値を出力する（Table 3、List 3参照)。

I/Oポートからアクセスする場合と異なり、1バイトずつ読む必要がないため、位置を指定するI/OポートA5Hへの出力は省略することができる。ただし、すべての全角文字がCGウィンドウに16×16ドットの形で現れるわけではない。上位バイトが2CH〜2FH(全角罫線特殊文字)、76H(ユーザー定義文字)、79H〜7CH（NEC拡張漢字）の全角文字は一度に左半分と右半分のどちらかしかCGウィンドウに現れない。このときは、左右の指定をI/OポートA5Hに対して行わなければならない（Table 4参照)。

8 CGウィンドウからデータを読み出す

CGウィンドウは、テキストアトリビュートVRAMの後ろにあるA4000H〜A4FFFHの4Kバイトの領域に割り当てられる。その中で実際に使われる部分はA4000H〜A401FHの32バイトのみで、それ以降は同じパターンの繰り返しとなる。

CGウィンドウに表れるパターンデータはFigure 2のような形式になっている。先頭から、全角文字のライン0の左、右、ライン1の左、右、ライン2の左、右…のようにして32バイトのデータが連続的に配置される。なお、ANK文字はVSYNC期間中でしか文字パターンを読み出せない。ここに示したANK文字のパターンはVSYNC期間中に現れるパターンである。

I/Oポートのアクセスと違い32バイトのパターンデータがメモリ上にあるため、連続的なメモリの読み出しのみでパターンデータを読むことができる。そのため、I/Oポートのアクセスに比べて高速である。

たとえば、全角漢字の'柏'(JIS漢字コード4770H)の文字パターンを得るには、I/OポートA1Hに漢字下位バイトデータ70Hを、I/OポートA3Hに漢字上位バイト47H−20Hを出力し、I/OポートA5Hに最上段左の位置データ20Hを出力する。そしてセグメントA400Hのオフセット0からパターンデータを読み出す。実際のコードではList 7のように16回のLODSW命令で全角文字1文字分のパターンデータを読み出す。

●全角文字のパターンデータのCGウィンドウ格納形式（その1）
上位バイトが2CH～2FH,76H,79H～7CHまでの全角文字を除く
例：'柏'JIS漢字コード 4770H

メモリアドレス	下位バイト								上位バイト								
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	
0A4000H	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	：1008H
0A4002H	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	：1008H
0A4004H	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	：2008H
0A4006H	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	：FF3EH
0A4008H	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	：8108H
0A400AH	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	1	：8118H
0A400CH	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	1	：811CH
0A400EH	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	1	：812AH
0A4010H	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	：8128H
0A4012H	0	0	1	0	1	0	0	0	1	1	1	1	1	1	1	1	：FF28H
0A4014H	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	1	：8148H
0A4016H	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	：8108H
0A4018H	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	：8108H
0A401AH	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	：8108H
0A401CH	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	：8108H
0A401EH	0	0	0	0	1	0	0	0	1	1	1	1	1	1	1	1	：FF08H

●全角文字のパターンデータのCGウィンドウ格納形式（その2）
上位バイトが2CH～2FH（罫線文字）,76H（外字）,79H～7CH（NEC拡張文字）の全角文字

例：'①'JIS漢字コード 2D21H
I/Oポート A5Hに 20Hを出力して左指定を行ったときのデータ

メモリアドレス	下位バイト								上位バイト								
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	
0A4000H									0	0	0	0	0	0	1	1	：03H
0A4002H									0	0	0	0	1	1	0	0	：0CH
0A4004H									0	0	0	1	0	0	0	0	：10H
0A4006H									0	0	1	0	0	0	0	0	：20H
0A4008H									0	0	1	0	0	0	0	1	：21H
0A400AH									0	1	0	0	0	0	1	0	：42H
0A400CH									0	1	0	0	0	0	0	0	：40H
0A400EH									0	1	0	0	0	0	0	0	：40H
0A4010H									0	1	0	0	0	0	0	0	：40H
0A4012H									0	1	0	0	0	0	0	0	：40H
0A4014H									0	1	0	0	0	0	0	0	：40H
0A4016H									0	0	1	0	0	0	0	0	：20H
0A4018H									0	0	1	0	0	0	0	1	：21H
0A401AH									0	0	0	1	0	0	0	0	：10H
0A401CH									0	0	0	0	1	1	0	0	：0CH
0A401EH									0	0	0	0	0	0	1	1	：03H

● 1バイトANKコードのCGウィンドウ格納形式
　ビットマップモードではアクセス不可

例：'A' ANKコード 41H

メモリアドレス	下位バイト D7 D6 D5 D4 D3 D2 D1 D0	上位バイト D15 D14 D13 D12 D11 D10 D9 D8	
0A4000H		0 0 0 0 0 0 0 0	: 00H
0A4002H		0 0 0 0 0 0 0 0	: 00H
0A4004H		0 0 0 0 1 0 0 0	: 08H
0A4006H		0 0 0 1 0 1 0 0	: 14H
0A4008H		0 0 1 0 0 0 1 0	: 22H
0A400AH		0 1 0 0 0 0 0 1	: 41H
0A400CH		0 1 0 0 0 0 0 1	: 41H
0A400EH		0 1 0 0 0 0 0 1	: 41H
0A4010H		0 1 0 0 0 0 0 1	: 41H
0A4012H		0 1 1 1 1 1 1 1	: 7FH
0A4014H		0 1 0 0 0 0 0 1	: 41H
0A4016H		0 1 0 0 0 0 0 1	: 41H
0A4018H		0 1 0 0 0 0 0 1	: 41H
0A401AH		0 0 0 0 0 0 0 0	: 00H
0A401CH		0 0 0 0 0 0 0 0	: 00H
0A401EH		0 0 0 0 0 0 0 0	: 00H

Figure 2　CG ウィンドウのパターンデータ格納状態

List 7　'柏'の文字パターンデータの読み出し

```
cld
mov     ax,0A400h           ; CG ウィンドウセグメント
mov     ds,ax               ;
mov     si,0                ; CG ウィンドウ先頭オフセット
mov     ax,4770H - 2000H    ; '柏' JIS 漢字コード 4770H
out     0A1h,al             ; 下位バイト出力　70H
xchg    ah,al
out     0A3h,al             ; 上位バイト出力 47H - 20H
lodsw                       ; CG ウィンドウからデータの読み出し
```

9 コードアクセスモードに変更する

I/O ポートから読み出すときと同様、パターン読み出し処理が終了した時点で、I/O ポート 68H に 0AH を出力してコードアクセスモードに戻し、漢字テキストが表示できるモードにする（Table 2、List 6 参照）。

10 グラフィック画面へ描画する

以上で 32 バイトの文字パターンデータを読んだため、つぎにそれをグラフィック画面に描画する。

I/O ポートによる読み出しの場合、全角文字をグラフィック画面に表示するには、パターンデータ位置を指定するための値を 20H から 2FH まで変化させながら 16 バイトのパターンデータを読み出し、そのデータによって文字の左半分を描画する。続けて位置を指定する値を 00H から 0FH まで変化させながら右半分の文字を描画する。

CG ウィンドウの場合は、16 ワードのメモリワード転送によって描画を行う。

描画するとき、GRCG の RMW モードを利用して 4 画面同時書き込みを行うと高速な描画が可能になる。これについては別節「GDC による 4 プレーン描画」を参照してほしい。

処理として一番大変なのがドット単位の位置への表示である。PC-9801 のグラフィックメモリマップは「グラフィック画面の基礎知識」の節で説明してあるように 1 ワードのデータが横に並ぶプレーン型である。そのため、縦ドット位置の指定は容易だが、横ドット位置の指定は 8 ドット単位（0、8、16、24...）でないとむずかしい。そこで横位置が 8 ドット単位ではない場合は、パターンデータ自体をシフトして変更する。その結果、横 16 ドットデータは最大 7 ドットシフトされるため、24 ドットデータとして描画する。

VRAM オフセットは、横座標を X、縦座標を Y とすると、Y×80＋X÷8 で求めることができる。このときに、X÷8 の余りの値がパターンデータを右にシフトする数となる。例を Figure 3 に示す。

```
・ドット位置描画の例

X＝14、Y＝3 のとき

        オフセットアドレス 3×80＋14÷8＝241

        ドットシフト数 14%8＝6

パターンデータ
        AL                 AH
1 1 1 1 1 1 1 1   1 1 1 1 1 1 1 1

                      ↓6ドット右シフトする
        AL                 AH                 DL
0 0 0 0 0 0 1 1   1 1 1 1 1 1 1 1   1 1 1 1 1 1 0 0

オフセット 241に AX をライトし、 243 に DLをライトする。
```

Figure 3　ドット描画位置の設定

▶ワンポイント

■データの加工

キャラクタジェネレータから読み込んだデータは、自由に加工することができる。加工したデータをグラフィック画面に描くことにより装飾に富んだ文字表示が可能となる。

たとえば、文字を太く表示することができる。太字のパターンデータは全角文字を1ドットシフトし、シフトしたデータと元のデータのORをとって作る。元のデータの例をFigure 4に示す。

	下位バイト								上位バイト								
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	: 1008H
1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	: 1008H
2	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	: 2008H
3	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	: FF3EH
4	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	: 8108H
5	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	1	: 8118H
6	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	1	: 811CH
7	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	1	: 812AH
8	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	: 8128H
9	0	0	1	0	1	0	0	0	1	1	1	1	1	1	1	1	: FF28H
10	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	1	: 8148H
11	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	: 8108H
12	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	: 8108H
13	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	: 8108H
14	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	: 8108H
15	0	0	0	0	1	0	0	0	1	1	1	1	1	1	1	1	: FF08H

Figure 4　JIS漢字コード4770H'柏'の文字パターン

この取り出したデータをラインごとにシフトさせ、元のデータとのORをとる。具体的に、Figure 4の第4ラインを太字のデータに変換する様子をFigure 5に示した。まず、第4ラインのデータを左に1ドットローテイトする。ただのシフトでは、D15のデータがD0に反映されない。そしてつぎに元のデータとシフトしたデータのORをとる。これを各ライン繰り返してデータを生成すると、Figure 6のように元の文字を太字に変換できる。

プログラムはList 8のようになる。

シフト前　　下位バイト　　　　上位バイト

0 0 0 0 1 0 0 0　1 0 0 0 0 0 0 1　：8108H

↓

シフト後

0 0 0 1 0 0 0 1　0 0 0 0 0 0 1 0　：0211H

↓

シフト前とシフト後のデータのORをとったデータ

0 0 0 1 1 0 0 1　1 0 0 0 0 0 1 1　：8319H

Figure 5　第4ラインの太字データ生成の様子

	L								H								
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_{09}	D_{08}	
LINE 0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	：3018H
LINE 1	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	：3018H
LINE 2	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	：6018H
LINE 3	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	：FF7FH
LINE 4	0	0	0	1	1	0	0	1	1	0	0	0	0	0	1	1	：8319H
LINE 5	0	0	1	1	1	0	0	1	1	0	0	0	0	0	1	1	：8339H
LINE 6	0	0	1	1	1	1	0	1	1	0	0	0	0	0	1	1	：833DH
LINE 7	0	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1	：837FH
LINE 8	0	1	1	1	1	0	0	1	1	0	0	0	0	0	1	1	：8379H
LINE 9	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	：FF79H
LINE 10	1	1	0	1	1	0	0	1	1	0	0	0	0	0	1	1	：83D9H
LINE 11	0	0	0	1	1	0	0	1	1	0	0	0	0	0	1	1	：8319H
LINE 12	0	0	0	1	1	0	0	1	1	0	0	0	0	0	1	1	：8319H
LINE 13	0	0	0	1	1	0	0	1	1	0	0	0	0	0	1	1	：8319H
LINE 14	0	0	0	1	1	0	0	1	1	0	0	0	0	0	1	1	：8319H
LINE 15	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	：FF19H

Figure 6　太字に変換したあとの'柏'のパターン

List 8　太字加工のルーチン

```
        cld
        mov     si,OFFSET DGROUP:kanji_data    ; 漢字パターンデータ格納アドレス
        mov     di,si
REPT KANJI_H                           ; KANJI_H = 16
        lodsw                          ; AX = パターンデータ
        mov     dx,ax
        rol     dx,1
        and     dx,0FEFF               ; bit8 をクリアする
        or      ax,dx
        stosw                          ; 変更されたパターンデータ格納
ENDM
```

■コードアクセスモードでの読み出し

本文でふれたように、ビットアクセスモードのほかにコードアクセスモードというモードがある。通常の、漢字テキストがディスプレイに表示されている状態がコードアクセスモードである。コードアクセスモードでの読み出しは、VSYNC を待つため速度は遅いが、データを読み出す間も文字画面が乱れたりしない。

キャラクタジェネレータは、テキスト画面表示が行われているときには絶えずハードウェアが読み出しを行っている。そのため、CPU がキャラクタ ROM データを読み出せるのは、文字表示を行っていない期間の中の VSYNC 信号が発生している期間のみである。

したがって、コードアクセスモードで読み出すには、まず VSYNC を検出する。「VSYNC 信号の検出」の節でも述べたように、VSYNC の期間をできるだけ長くとるために、VSYNC の発生した直後を検出する。数バイト(全角漢字で 32 バイト、ANK 文字で 16 バイト)の読み出しを連続で行うためには時間が必要で、VSYNC 期間の途中から読み出しを始めた場合には、VSYNC 期間中に読み出しが終了しない可能性があるからである。VSYNC 期間で読み出しが終わらなかった場合は無効なデータ(実際は FFH)になってしまう。

なお、ANK 文字のパターンデータはコードアクセスモードで読み出さなくてはならない。そのため実際のプログラムでは、起動時などに ANK 文字だけまとめてパターンデータを読み出してメモリに置いておき、表示するときにはそれを使うという方法が用いられる。

■背景の消去

GRCG の RMW モードはすでにグラフィック画面にあるデータとの OR であるため、表示されたデータの上に文字を描画すると前のグラフィックデータと重なった表示状態になってしまう。

テキスト表示のときにはそのようなことはないため見逃しやすいことであるが、グラフィック文字描画をする際は、すでに描かれているグラフィック画面データを領域塗り潰しルーチンなどで消去する必要がある。

■CGウィンドウの成り立ち

CG ウィンドウによるアクセスにも I/O ポートアクセスと同様にコードアクセスモードとドットアクセスモードがあるが、この 2 つのモードを持つということは、CG ウィンドウはキャラクタジェネレータのイメージをそのままこのアドレスに表しているものと考えられる。

ライブラリ

KanjiGputc

解説　全角文字をグラフィック画面の指定したドット位置に表示する。引数COLORのbit 8を1にすると、太文字が描画される。

void **KanjiGputc**(unsigned int *KANJI*,int *XP*,int *YP*,int *COLOR*)

KANJI シフトJIS漢字コードの上位バイトと下位バイトを入れ替えたもの
XP 表示 横座標 0〜639−16
YP 表示 縦座標 0〜400−16
COLOR 表示カラー
bit8 0：普通文字
1：太文字

高速化を図るためマシンの種別を判断し最適な処理を実行する。
グラフィック画面への描画なので、その利点を利用してアナログ16色表示、太字処理、ドット単位の描画位置の指定が行えるようになっている。ただし全角文字(上位バイトが2CH〜2FH、76H、79H〜7CHの全角文字は除く）の表示のみで1バイトANK文字及び2バイト系半角文字は表示できない。
文字パターンデータの読み出しは、ドットアクセスモードで行っているため、漢字テキストを表示した状態でこの関数を実行すると漢字テキスト表示が一瞬乱れることがある。
引数KANJIの仕様はTurbo C++、Borland C++ではKanji Gputc(‘柏’、100、100、15）といった呼び出し方をするためのものである。Quick CやMS-Cでは、定数をそのまま書けばよい。
なお、この関数はVMタイプ、EGCタイプのPC-9801のみで動作する。

戻り値　なし

サンプル　GDEMO.C

G R C G 編

GRCGの基礎知識

GRCG（グラフィックチャージャ）は、複数プレーン同時書き込み、読み出しを可能にするハードウェアである。GRCGには8ビットのタイルレジスタが4プレーン分用意されており、この内容とCPUからのデータによってグラフィック画面に書き込みが行われる。

GRCGの動作モードには、書き込みのためのTDWモードとRMWモード、読み出しのためのTCRモードの3つのモードがある。ここでは、この3つのモードについて解説し、TDWモードを使ったグラフィック画面消去関数、RMWモードを使った塗り潰し四角形の描画関数を紹介する。

GRCGの概要

GRCGは、VMタイプのPC-9801から搭載された、ハードウェア的にグラフィックの高速描画をサポートする機構である。GRCGは、グラフィックの青、赤、緑、拡張の4プレーンを同時に読み出したり書き込んだりする機能を提供する。とくに同じ色やパターンの書き込みに威力を発揮し、グラフィック処理速度を大幅にアップする。最近のグラフィックを扱うソフトウェアの多くがこの機能を利用しており、アナログ表示と合わせて、グラフィック描画処理が多いソフトウェア、とくにゲームでは必須の機能である。

なお、一部のPC-9801に搭載されているEGCは、GRCGをさらに強化したグラフィック用のアクセラレータで、GRCGの互換のモードを持っている。

GRCGのI/Oポート

GRCGは、CPUとグラフィックVRAMの間に存在する。ただし、I/Oポートを操作してGRCGを明示的に有効にしなければ、GRCGのないPC-9801のグラフィック部とまったく同じである。

GRCGの動作モードを設定するのはTable 1に示すI/Oポート7CHで、モードレジスタと呼ばれる。モードレジスタは、GRCGの有効や無効、あるいは、どのプレーンに対してGRCGが有効かを設定できる。GRCGを利用するときには、はじめにこのポートを操作する必要がある。

GRCGの重要な機構が、タイルレジスタである。GRCG経由のグラフィックVRAMへの書き込みでは、必ずこのタイルレジスタを通ったデータが書き込まれる。タイルレジスタの設定は、I/Oポート7EHを通して行う。

Table 1　GRCGのI/Oポート

I/Oポート	Read/Write	意　味
7CH	Write	モードレジスタ ビット　意　味 bit7　CG（GRCG mode set） GRCGを有効にする 値　意　味 0　GRCG無効（標準のVRAMアクセス） 1　GRCG有効 bit6　RMW（RMW mode set） RMWモードにする 値　意　味 0　TDW mode（VRAM write時） TCR mode（VRAM read時） 1　RMW mode（VRAM write only） bit5、4　常に0にセットする（ノーマルモード時） bit3　P3（Plane 3 enanle） プレーン3へのアクセスを有効にする bit2　P2（Plane 2 enanle） プレーン2へのアクセスを有効にする bit1　P1（Plane 1 enanle） プレーン1へのアクセスを有効にする bit0　P0（Plane 0 enanle） プレーン0へのアクセスを有効にする
7EH	Write	タイルレジスタ

GRCGのモード設定

GRCGを利用するには、まずモードレジスタ（I/Oポート7CH）を操作する必要がある。**モードレジスタ（I/Oポート7CH）のbit7を1にしてGRCGによる動作をONにし、bit6の値を設定してモードを設定する。**それと同時にbit0～3の4ビットでアクセスプレーンを設定する（このビットは0で有効）。

List 1の例は、GRCGをRMWモードで有効にし、4プレーンすべてに対してアクセスを行う設定である。

List 1　GRCGをRMWモードで有効にするプログラム

```
mov     al,0C0h             ; GRCG RMW モード
out     7Ch,al
```

I/Oポートから設定したモードの内容は、システム共通域に格納しておくべきである。こうすれば、グラフィック描画を行う割り込みプログラムとの共存が可能となる（Table 2参照）。

Table 2　システム共通域

アドレス	意　味	
0000:0495H	EGC/GRCGのモードレジスタの内容（GRAPH_CHG）	
	bit7	EGC/GRCGの利用状況
	0	使用していない
	1	使用中

タイルレジスタの設定

GRCGにはタイルレジスタと呼ばれる8ビットのレジスタが各プレーンごとに用意されている。GRCGはこのタイルレジスタの内容に従ってグラフィックVRAMに対する書き込みと読み出しを行う。

モードレジスタにデータを出力してGRCGを有効にすると、I/Oポート7EHはタイルレジスタ0となり、プレーン0に対応するタイルデータを出力できる。続けてデータを出力するとI/Oポート7EHはタイルレジスタ1に変わり、続けて出力を行うとタイルレジスタ2、3と変化し、4つのタイルレジスタの設定を行うことができる。

このため、タイルレジスタ2だけ設定するといったことはできない。つまり、タイルレジスタは連続して設定する必要があり、その間にマウス割り込みによる描画などが行われないように割り込みを禁止する必要がある。

List 2は、タイルレジスタに値を設定する例である。

タイルレジスタを設定する際、その内容をシステム共通域に格納することにより、マウスドライバなどのハードウェア割り込みでグラフィック描画を行うソウトウェアとの共存が可能となる。マウスドライバのような、割り込みからGRCGの設定を変更しグラフィック描画を行うソフトウェアにおいては、このシステム共通域を参照し、GRCGの設定を割り込み前の状態に戻してから割り込みから復帰するよう作成する必要がある（Table 3参照）。

List 2　タイルレジスタの設定

```
sub     ax,ax           ; ax = 0
mov     es,ax
pushf
cli
mov     al,0FFh
out     7Ch,al          ; タイルレジスタ0に出力
```

```
mov     es:[0496h],al
mov     al,00h
out     7Ch,al                ; タイルレジスタ1に出力
mov     es:[0497h],al
mov     al,0FFh
out     7Ch,al                ; タイルレジスタ2に出力
mov     es:[0498h],al
mov     al,00h
out     7Ch,al                ; タイルレジスタ3に出力
mov     es:[0499h],al
popf
```

Table 3　システム共通域

アドレス	意　味
0000:0496H～0499H	GC/EGCのタイルレジスタの設定値（GRAPH_TAL） マウスドライバが割り込みルーチン内で破壊したGCのレジスタを処理終了時に復旧するために用いる。

GRCGの3つのモード

GRCGには、グラフィックVRAMへの書き込みのためのTDWモードとRMWモード、読み出しのためのTCRモードの合計3つのモードがある。

TDWモードは、タイルレジスタの内容をCPUデータの内容とは無関係にグラフィックVRAMへ書き込む。RMWモードは、タイルレジスタの内容とCPUデータの内容とのANDをとり、その値をグラフィックVRAMへ書き込む。読み出しのためのTCRモードは、グラフィックVRAMの内容をタイルレジスタの内容と照合し、一致したデータをCPUに渡すものである。以下、この3つのモードについて解説する。

TDWモードによる書き込み

GRCCをTDWモードにしてタイルレジスタの設定をしたときに、CPUがVRAMのアドレス（プレーン0のアドレスA800:0000H～7FFFH）に対して書き込みを行うと、CPUからの書き込みデータに関係なく（アドレスには関係する）4つのタイルレジスタの内容が各プレーンに書き込まれる（List 3参照）。

List 3　TDW モードによる書き込みの例

```
mov     ax,0A000h
mov     es,ax
mov     es:[0000h],al              ; al の値はなんでもよい
```

たとえば、タイルレジスタ 0 に 11H、タイルレジスタ 1 に 22H、タイルレジスタ 2 に 33H、タイルレジスタ 3 に 44H のデータを設定し、グラフィック VRAM の A800:0000H に対して書き込みを行うと、ディスプレイの左上端から 0000B、1000B、0110B、0101B、0000B、1000B、0110B、0101B の色番号のドットが表示されることになる(Figure 1 参照)。このドットは、アナログモードで標準のパレットにおいては、黒、暗グレー、暗黄色、暗シアン…になる。つまり、グラフィックプレーン 0 の内容が色番号の bit 0 に対応し、4 つのプレーンの内容により 4 ビットの色番号が決定する。

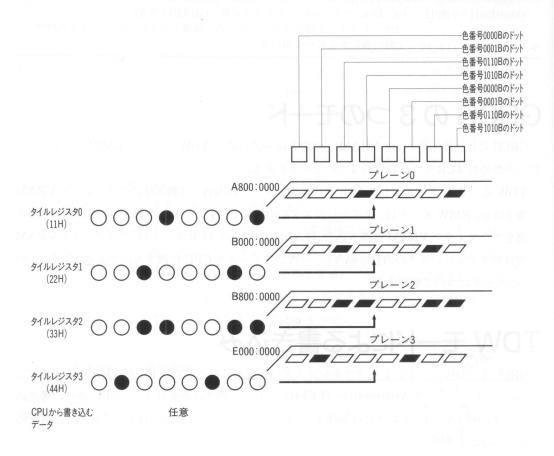



Figure 1　TDW モードで書き込まれる値

このようにCPUからグラフィックプレーン0へ書き込むだけで4プレーンの描画が行われる。CPUのデータとは無関係に書き込みが行われるため、このモードは画面消去などに使われるが、あまり使いみちは広くない。

CPUデータが1ワード（2バイト書き込み）のときタイルレジスタは2バイト（同じ1バイトデータが上位バイトと下位バイトに）に拡張される。

RMWモードによる書き込み

GRCGをRMWモードにすると、CPUから書き込むデータと各タイルレジスタのデータのANDをとった結果を各グラフィックプレーンに書き込む。CPUデータが1のとき対応するタイルレジスタのビットの内容が各プレーンに書き込まれ、CPUデータが0のビットはすでにプレーン上にあるデータが残される（List 4参照）。

List 4　RMWモードによる書き込みの例

```
mov     ax,0A000h
mov     es,ax
mov     al,0F0h
mov     es:[0000h],al
```

たとえば、グラフィックVRAM上に色番号15（デフォルトでは白）の8ドットのデータがあるとき、そのアドレスにCPUからデータF0Hを書き込むと、プレーン0には01011111B、プレーン1には000011111B、プレーン2には01011111B、プレーン3には00001111Bのデータが書き込まれる。その結果ディスプレイ上に表示されるものは、4ドットの色番号5と0の破線（シアンの破線）と、その右に4ドットの長さの白線となる（Figure 2参照）。

TDWモードのときと同様に、CPUの書き込みデータが1ワード（2バイト書き込み）のときタイルレジスタは2バイト（同じ1バイトデータが上位バイトと下位バイトに）に拡張される。

このRMWモードはGRCGのモードの中でもっとも実用度の高いもので、同じ色にとどまらず同じパターン図形の書き込みにも多く使われる。すでにあるグラフィックデータとのOR論理演算をしながら4プレーン同時書き込みを行うため、応用範囲も広く、ゲームなどのグラフィック表示処理が多くを占めるソフトウェアでは必須のモードである。

ただし、このモードは書き込み専用であり、このモードではけしてVRAMから読み出しを行ってはならない。RMWモードのときに読み出しを行うとハングアップする恐れがある。グラフィックVRAMをワークエリアとして使うような割り込み型のソフトウェアでは、とくに注意を要する。

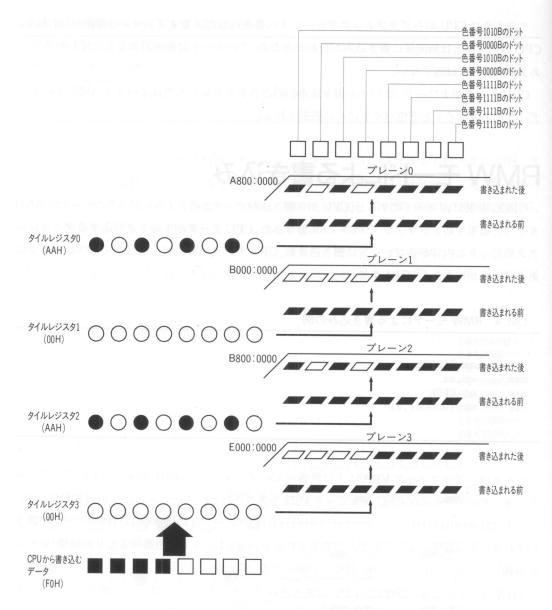



Figure 2　RMW モードでの書き込み例

TCR モードによる読み出し

GRCGをTCRモードにしてCPUがグラフィックVRAM（プレーン0）から読み出しを行うと、プレーンとそのタイルレジスタとの一致したデータを各プレーン毎に作成し、4つのプレーンのデータをANDした値をCPUに渡す。

タイルレジスタに設定するデータは、色データの検出においては、FFHか00Hとする。たとえ

ば、タイルレジスタ0にFFH、タイルレジスタ1に00H、タイルレジスタ2にFFH、タイルレジスタ3に00Hのデータを設定し、グラフィックVRAMのA800:0000Hに対して読み出しを行うと、Table 4、Table 5のどちらのグラフィックデータでも、CPUには01Hが読み込まれることになる（Figure 3参照）。

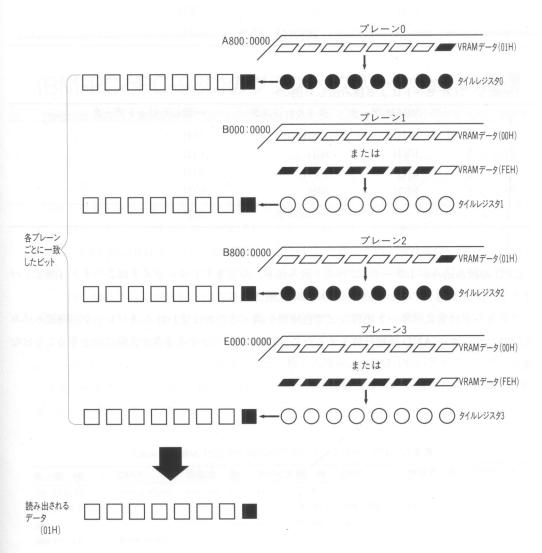



Figure 3　TCRモードによる読み出し

Table 4　TCR モードによる読み出し（例 1）

	VRAM データ	タイルレジスタ	一致したビットデータ
プレーン 0	01H	FFH	01H
プレーン 1	00H	00H	FFH
プレーン 2	01H	FFH	01H
プレーン 3	00H	00H	FFH
4 プレーンの一致したビットデータの AND 値			01H

Table 5　TCR モードによる読み出し（例 2）

	VRAM データ	タイルレジスタ	一致したビットデータ
プレーン 0	01H	FFH	01H
プレーン 1	FEH	00H	01H
プレーン 2	01H	FFH	01H
プレーン 3	FEH	00H	01H
4 プレーンの一致したビットデータの AND 値			01H

CPU の読み込みが 1 ワード（2 バイト読み込み）のときタイルレジスタは 2 バイト（同じ 1 バイトデータが上位バイトと下位バイトに）に拡張される。

このモードは主にペイント処理などで色境界を調べるために使われる。4 プレーン同時読み込みと同時にデータの AND 論理計算をするため、境界をチェックする速度が大幅に向上することになる。

COLUMN

IBM-PCとPC-9801のグラフィックハードウェアの変遷

1982年当時のPC-9801のグラフィック性能は、IBM-PCと比べてはるかに優れたもので640×400ドットで漢字表示が可能なものであった。1985年にIBM-PCとPC-9801は大きなグラフィックの性能向上が行われ、IBM-PCは640×350 16色表示のEGAに、PC-9801はドット数は同じままで16色表示となりGRCGが搭載され、この時点でグラフィック性能は同程度になった。

GRCGは4プレーン同時書き込みと読み込みが可能だが、EGAでもほぼ同じことができ、さらに指定プレーンの読み込みが可能であるため、GRCGよりやや優れている。

そして1987年にIBM-PCはVGAを搭載し、ドット数の面でもPC-9801を追い抜いた。VGAは表示できる色数も多く、かなり複雑なグラフィック描画制御が可能となっている。それに比べてPC-9801はEGCが搭載されたのみで、その後の発展はほとんどなく、PC-9801シリーズとは少し異なった流れのPC-H98でグラフィックを強化したが、その機能を使ったアプリケーションはほとんど存在しない状況である。

このPC-9801のグラフィックが改善されない最大の理由は、IBM-PCがグラフィックボードの交換が可能であるのに対し、PC-9801はグラフィック機能が本体に内蔵されて、交換不可能なことにあるといえるだろう。

Table A　IBM-PCとPC-9801のグラフィックハードウェアの変遷

年	機械	CPU	解像度	色	デバイス	機械	CPU	解像度	色	デバイス
1981	IBM-PC	8088/4.8MHz	640×200	4色	CGA	——	——	——	——	——
1982	PC-XT	↑	↑	↑	↑	PC-9801	8086/8MHz	640×400	8色	GDC
1983	——	——	——	——	——	PC-9801F	8086/8MHz	↑	↑	↑
1984	PC-AT	80286/6MHz	↑	↑	↑	——	——	——	——	——
1985	PC-AT	80286/8MHz	640×350	16色	EGA	PC-9801VM	V30/8MHz	↑	↑	GDC、GRCG
1986	——	——	——	——	——	PC-9801VX	80286/8MHz	↑	↑	GDC、EGC
1987	PS/2	80386DX/16MHz	640×480	256色	VGA	——	——	——	——	——
1988	——	——	——	——	——	——	——	——	——	——
1989	——	——	——	——	——	——	——	——	——	——
1990	IBM-PC互換機	80386DX/33MHz	800×600	256色	SuperVGA	PC-H98/70	80386DX/33MHz	1120×750	256色	ACDC、E^{2}GC
1991	IBM-PC互換機	80486/33MHz	1024×768	256色	XGA	PC-H98/90	80486SX/25MHz	↑	↑	↑

GRCGによるグラフィック画面の消去

GRCGのTDWモードによる4プレーン同時書き込みの機能を利用すると、指定した色でグラフィク画面全体を高速に塗りつぶすことができる。ここでは、タイルレジスタに指定の色番号を設定し、グラフィックプレーン0へCPUから連続で書き込み、高速に画面をクリアする方法について解説する。

▶ポイント

- GRCGをTDWモードで有効にする。
- CPUのローテイト命令を使用して色番号を高速にタイルレジスタに設定する。
- グラフィックプレーン0に対して、CPUから連続して書き込みを行う。
- 処理が終わったらGRCGを無効にする。

▶プログラミングテクニック

1 GRCGをTDWモードに設定する

GRCGをTDWにすると、グラフィックVRAMに書き込む値とは関係なくグラフィック画面に同じ色を塗ることができる。これを使ってグラフィック画面を消去する。

まず最初にI/Oポート7CHのbit7を1にしてGRCGを有効にし,bit6を0にしてTDWモード（読み込みではTCRモード）に設定する。

I/Oポート7CHのbit0、1、2、3はそれぞれプレーン0、1、2、3へのアクセスビットであり、0で有効となる（「GRCGの基礎知識」のTable 1参照)。ここでは、全プレーンへのアクセスを有効にする。プログラムはList 1のようになる。

この設定のあと、CPUからグラフィックVRAMへのアクセスはすべて、4プレーン同時アクセスとなる。書き込みにおいては、タイルレジスタの内容がCPUによってアクセスされたグラフィックVRAMアドレスに書き込まれる。読み出しにおいては、タイルレジスタと一致したグラフィックVRAMデータが読み込まれる。

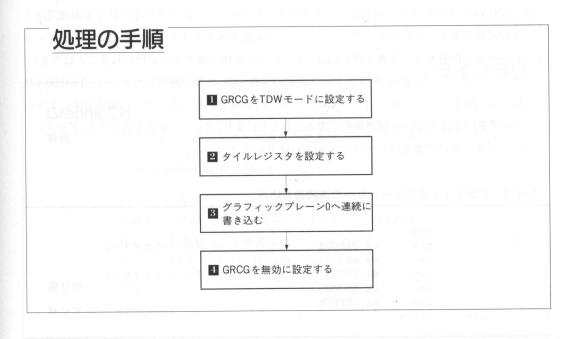
List 1　GRCG を TDW モードに設定

```
pushf
cli                     ; 割り込み禁止
mov     al,080h         ; TDW or TCR モードプレーン 0,1,2,3 有効
out     7Ch,al          ; GRCG セット
mov     dx,0
mov     es,dx
mov     es:[0495h],al   ; GRCG モードをシステム共通域にセット
popf
```

2 タイルレジスタを設定する

TDW モードでは、4 つのタイルレジスタの内容で、書き込みの内容を決定する。指定した色番号で書き込むためには、I/O ポート 7EH を使って、データを書き込むプレーンに FFH を、データを消去するプレーンに 00H をセットする。たとえば色番号 5 の例を、Table 1 に示す。

Table 1　色番号 5 を書く場合のタイルレジスタの設定

タイルレジスタの番号	対応するプレーン	設定する値
タイルレジスタ 0	プレーン 0	FFH
タイルレジスタ 1	プレーン 1	00H
タイルレジスタ 2	プレーン 2	FFH
タイルレジスタ 3	プレーン 3	00H

この設定を行うのに単純な方法を用いると、16色の場合で、16個の判断処理と16個のデータテーブルが必要になる。しかし、このような処理ではスピードが遅くなってしまう。

そこで、指定された色番号(ALレジスタ)をローテイトして、最下位ビットを最上位ビットに移動し、CBW命令(ALレジスタの内容を符号付きでAXレジスタに拡張する)でAHレジスタに00HかFFHかを作成し、タイルレジスタとシステム共通域にセットする。この方法を使うと、かなり高速にタイルレジスタに値を設定できる。タイルレジスタを設定するプログラムはList 2のようになる。

List 2　タイルレジスタの設定

```
;----------- al = 色番号
gds_1:          ror     al,1            ; 色番号をローテイト
                cbw                     ; bit = 1 のとき ah = FFH
                xchg    ah,al           ; bit = 0 のとき ah = 0 となる
                out     07Eh,al         ; タイルレジスタにデータセット
                stosb                   ; システム共通域へデータ格納
                xchg    ah,al
                loop    gds_1
```

3 グラフィックプレーン0へ連続に書き込む

GRCGを使ったグラフィックVRAMへのアクセスは、グラフィックプレーン0に対してのみ行う。GRCGをTDWモードに設定してあるので、1プレーンへのアクセスだけで4画面同時アクセスが可能である。これによって、グラフィック画面全体をクリアするためには、グラフィックプレーン0の内容をすべて書き換えれば、4プレーン全体の書き込みが行われることになる。

グラフィックプレーン0は「グラフィック画面の基礎知識」の節で説明したように、A800:0000Hから始まる32000バイト（80バイト×400ライン）の領域である。そこでこの領域全体に1ワードのデータFFFFHを16000個連続的に書き込むことにより、プレーン全体を消去する。グラフィックプレーン0へ連続で書き込むプログラムはList 3のようになる。

List 3　グラフィックプレーン0への連続書き込み

```
                cld
                mov     ax,0A800h               ; プレーン0セグメント
                mov     es,ax
                mov     di,0000h                ; プレーン0オフセット
                mov     cx,16000
                mov     ax,0FFFFh
        rep     stosw
```

4 GRCGを無効に設定する

GRCGによる処理が終了したら、GRCGを無効にして従来のグラフィックと同じ状態に戻す。続けてGRCGを利用するときは、この処理は必要ない。

GRCGを無効にするには、I/Oポート7CHのbit7を0にする。また、GRCGの有効の処理と同様にその状態をシステム共通域に格納する。このとき、処理が終了するまで割り込み禁止にする。List 4にプログラムを示す。

List 4　GRCGを無効に設定

```
pushf
cli                             ; 割り込み禁止
mov     al,00h                  ; GRCGを無効に
out     7Ch,al                  ; GRCGセット
mov     dx,0
mov     es,dx
mov     es:[0495h],al           ; GRCGモードをシステム共通域にセット
popf
```

この設定以降、CPUによるグラフィックVRAMへのアクセスは、通常どおりプレーンごと別々のアクセスとなる。

ライブラリ

GraphicCls

解説　グラフィック画面全体を指定した色番号で消去する。書式はつぎのとおり。

void **GraphicCls**(int *Color*)

Color　色番号

GRCGのTDWモードを利用して4プレーン同時書き込みをすることで、指定した色番号による高速なグラフィック画面クリアを実現している。VMタイプとEGCタイプのPC-9801でのみ動作する。

戻り値　なし

サンプル　GDEMO.C

GRCGによる塗り潰し四角形の描画

GRCGのRMWモードによる4プレーン同時書き込みの機能を利用して、グラフィック画面に塗り潰し四角形を描画する。この機能はグラフィックLIOにも用意されており、最新のPC-9800シリーズではLIOを利用しても、かなり高速に描画を行うことができる。ここでは、4プレーン同時描画以外にも高速化のためのテクニックを解説し、塗り潰し四角形の超高速描画の関数を作成する。

▶ポイント

- GRCGをRMWモードで有効にし、書き込むCPUデータとタイルレジスタとのANDをとったデータを描画するように設定する。
- 指定の色番号を高速にタイルレジスタに設定する。
- 左端と右端におけるワード単位境界からはずれるビットデータを求める。
- 左端から右端までの書き込むワード数を求める。
- グラフィックプレーン0に対して、左端から右端までの水平線を縦ライン数書き込む。

▶プログラミングテクニック

1 GRCGをRMWモードに設定する

「GRCGによるグラフィック画面の消去」の節で示したように、I/Oポート7CHのbit7を1にしてGRCGを有効にし、bit6を1にしてRMWモードに設定する。I/Oポート7CHのbit0、1、2、3は、それぞれプレーン0、1、2、3へのアクセス有効ビットであり、0で有効となる。ここでは当然、全プレーンを有効にする。プログラムをList 1に示す。

List 1　GRCGをRMWモードに設定

```
pushf
cli                 ; 割り込み禁止
mov al,0C0h         ; RMW モード プレーン 0,1,2,3 有効
out 7Ch,al          ; GRCG セット
mov dx,0
mov es,dx
mov es:[0495h],al   ; GRCG モードをシステム共通域にセット
popf
```

2 タイルレジスタを設定する

GRCGを有効にしたら、つぎはタイルレジスタを設定する。描画したい色に応じて、タイルのパターンを決定する。具体的な方法は、「GRCGによるグラフィック画面の消去」の節の 2 と同じである。詳細はそちらを参照して欲しい。

3 水平線描画のための両端のビットデータを決定する

グラフィック画面への四角形の描画は、ある一定の長さの水平な直線を、縦に必要なライン数だけ繰り返し描画することである。四角形の左端のX座標が0、16、32…になっていて、右端のX座標が15、31、47…のように、きちんとワード境界になっていれば、REP STOSW命令でFFFFHのデータを書き込めばよいが、それ以外のX座標ではワード境界からはずれる左右の半端なビットパターンを求める必要がある（Figure 1参照）。

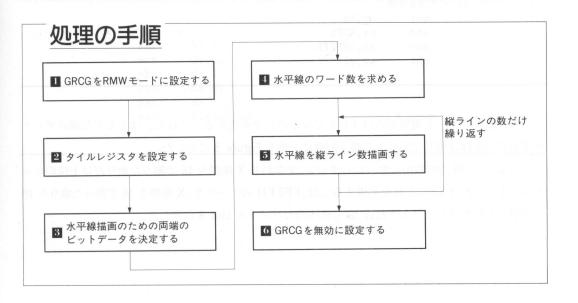
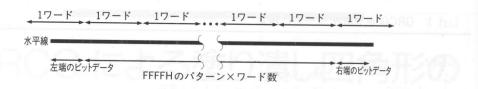



Figure 1　水平線の構造

まず、左端のデータを求めることを考える。例としてFigure 2のような3つのビットパターンがあったとする。

Figure 2　左端の例

このデータをX座標から求めるには、X座標を16で割る。すると余りは、ちょうどFigure 2の破線の部分になるので、その分だけFFFFHのデータ（つまりすべてが1のデータ）を右にシフトさせればよい。具体的なリストをList 2に示す。

List 2　左端のビットデータを求めるプログラム

```
;---------- bx = 左端X座標
        mov     cx,bx
        and     cx,00Fh
        mov     ax,0FFFFh
        shr     ax,cl
```

つぎに右端のワード境界からはずれるビットデータを求める。これも、ほとんど左端のデータを求めるのとほぼ同じである。左端と同様に一例をFigure 3に示した。

このように、16の倍数から余ったパターン、すなわちX座標を16で割った余りだけ上位のビットから1にすればよい。これを実現するには、FFFFHのデータを、X座標を16で割った余りを16から引いた分だけ左シフトすればよい。具体的なリストをList 3に示した。

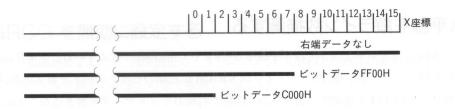



Figure 3　右端のデータ例

List 3　右端のビットパターンを求める

```
;---------- dx = 右端X座標
            mov     cx,dx
            and     cx,000Fh
            sub     cx,0010h
            neg     cx
            mov     ax,0FFFFh
            shl     ax,cl
```

4 水平線のワード数を求める

右端のビットデータと左端のビットデータを取り除くと、四角形を構成する水平線の長さは16の倍数となる。これを16で割ることにより水平線を描画するためのワード数を求められる。始点のX座標と終点のX座標がわかっていればList 4のプログラムで求めることができる。

List 4　ワード数を求める

```
;---------- bx = x1  dx = x2
            shr     bx,4
            mov     di,bx
            shr     dx,4
            sub     dx,di
            dec     dx
            mov     [lndx],dx                    ; ワード数を格納
```

5 水平線を縦ライン数描画する

2 でも紹介したとおり、ここでは水平線を3つのパートにわけて考えている。つまり、塗り潰し四角形を描くための水平線は、グラフィック VRAM に左端のビットデータを書き込み、続けてワード数分の FFFFH を連続的に書き込み、終端で右端のビットデータを書き込むことにより描画することになる。ワード数が1—つまり水平像が1ワードの12以内のときは、左端ビットデータと右端ビットデータの AND をとったデータを書き込むことになる（Figure 4 参照）。

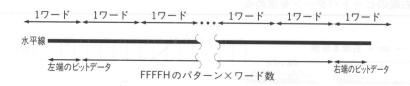



Figure 4　水平線の分解図

この水平線（すべて同じ長さ）を縦ライン数分連続的に描画することにより塗り潰し四角形となる。このとき高速化を実現するために共通のビットデータ等は、できるだけ CPU のレジスタに保持して描画を行う。プログラムを List 5 に示す。

List 5　四角形を描く

```
            mov     bx,[xpats]      ; 左端ビットデータ
            mov     dx,[xpate]      ; 右端ビットデータ
            mov     ax,0FFFFh       ; ワード書き込みのデータ
boxf_01:
            push    cx              ; cx = 縦ライン数
            push    di
            mov     cx,[lndx]       ; 水平線ワード数
            mov     es:[di],bx
            add     di,2
     rep    stosw
            mov     es:[di],dx
            pop     di
            add     di,80
            pop     cx
            loop    boxf_01
```

6 GRCGを無効に設定する

四角形の描画が終了したら、画面消去のときと同じようにGRCGを無効に戻す。詳細は、「GRCGによるグラフィック画面の消去」の節を参照して欲しい。

▶ワンポイント

■バイト境界による描画

左右端のビットデータはワード境界で求めたが、バイト境界でも可能である。バイト境界で求めるとビットデータの種類も少なくなり演算も楽である。しかし、PC-9801は16ビットのCPUを持ちグラフィックVRAMとの接続も16ビットバスであるため、ワード境界で描画を行う方がスピードの点で有利である。

ライブラリ

GraphicBoxf

解説

グラフィック画面にGRCGを利用して塗り潰し四角形を描画する。書式はつぎのとおり。

void **GraphicBoxf**(int *X1*,int *Y1*,int *X2*,int *Y2*,int *Color*)

X1,Y1	左上点座標
X2,Y2	右下点座標
Color	色番号

GRCGのRMWモードを利用して4プレーン同時に書き込むことで、高速な描画を実現している。VMタイプとEGCタイプのPC-9801でのみ動作する。

戻り値　なし

サンプル　GDEMO.C

ＥＧＣ編

EGCの基礎知識

EGCを搭載したPC-9801では、グラフィックの高速処理を実現するためのEGC(Enhanced Graphic Charger) が搭載されている。

EGCには、GRCG互換モードと拡張モードとがある。この章では、GRCG互換モードと拡張モードの一部の機能について紹介し、その操作方法を解説することにする。ただし、EGCについては資料がほとんどないため、この章の内容は筆者の実験と調査によるものであり、EGCを搭載した機種でも異なる動作をする可能性がある。

EGCの特徴

EGCには以下の特徴がある。

- GRCG互換モードを持つ。
- 4グラフィックプレーン同時制御が可能
- グラフィックプレーン同時処理におけるラスタオペレーションが可能
- グラフィックVRAMのブロック移動における、ビットシフト機能を有する

それぞれの機能が、グラフィックを処理するためには非常に有効であり、CPUのみで処理するのに比べ大幅なスピードアップを図ることができる。ただし、このEGCの機能の詳細は公開されていないことや、現在発売されているPC-9801のなかにもEGCが搭載されていないものがあるため、この高機能なEGCを使っていないソフトウェアがほとんどである。

EGCのI/Oポート

EGCを制御するI/Oポートには、6AH、04A0H、04A2H、04A4H、04A6H、04A8H、04AAH、04ACH、04AEHがある。詳細はつぎのとおりである。

■I/Oポート6AH

I/Oポート6AHは、EGCの動作モードを設定するI/Oポートである。詳細は、Table 1のとおり。

Table 1　拡張モードレジスタのI/Oポート

I/Oポート	Read/Write	意　味
6AH	Write	拡張モードレジスタ モード変更可能／不可を設定するフリップフロップ 値　意　味 00000101B　EGC 拡張モード 00000100B　GRCG 互換モード 00000111B　モード変更可能 00000110B　モード変更不可

I/Oポート6AHに07Hを出力することでモード変更可能にし、そのあとI/Oポート6AHへ05Hまたは04Hを出力することで、EGC拡張モードやGRCG互換モードへの変更が可能となる。EGC本来の能力を発揮するのがEGC拡張モードである。以下で説明するのはEGC拡張モードにおけるI/Oポートである。

■I/Oポート04A0H

I/Oポート04A0Hは、EGCのアクセスプレーンを決定するI/Oポートである。詳細はFigure 1のとおり。

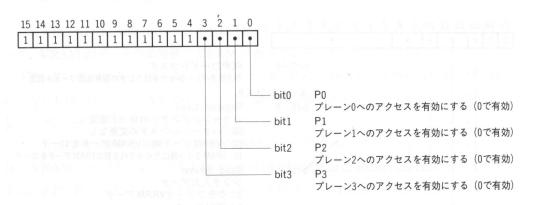



Figure 1　アクセスプレーン

■I/Oポート04A2H

I/Oポート04A2Hは、フォアグラウンドカラー（FGC）とバックグラウンドカラー（BGC）のON/OFFとリードプレーンを決定するI/Oポートである。詳細はFigure 2のとおり。

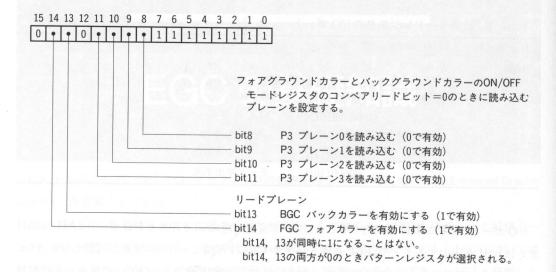



Figure 2 リードプレーン、フォアグラウンドカラーとバックグラウンドカラーの設定

■I/O ポート 04A4H

I/O ポート 04A4H は EGC の動作モードと ROP（ラスタオペレーション）を設定するポートである。詳細は Figure 3 のとおり。

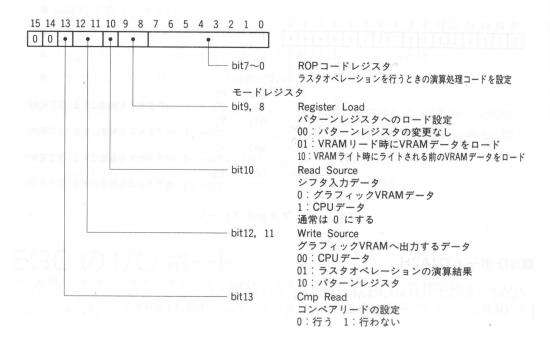



Figure 3 モードレジスタ、ROP コマンドレジスタ

ラスタオペレーションとは、CPUのデータとVRAM上のデータ、そしてパターンレジスタのデータの間の論理演算のことである。GRCGでは、2種類の演算(RMWモードとTDWモード)しか指定できなかったが、EGCでは8ビット――つまり256種類の描画方法が指定できる。このラスタオペレーションを指定するのがROPコードである。

ROPコードと実際のラスタオペレーションの間にはつぎのような関係がある。 ROPコードはFigure 4に示すとおり8ビットあり、それぞれのビットをR7〜R0とする。

7	6	5	4	3	2	1	0
R7	R6	R5	R4	R3	R2	R1	R0

Figure 4 ROPコードの8ビット

また、入力CPUのデータをS (Source)、VRAM上のデータをD (Destination) とし、パターンレジスタのデータをP (Pattern) とする。各々のデータは1か0の値を持つとする。すると、R7〜R0とS、D、Pでつぎのような論理演算が行われて、VRAM上に書き込まれる(書き込まれるデータをRとする)。なお、・はANDを、＋はORを、$\overline{X}$はNOTを表す。

$$R = R7 \cdot (P \cdot S \cdot D) + R6 \cdot (\overline{P} \cdot S \cdot D) + R5 \cdot (P \cdot S \cdot \overline{D}) + R4 \cdot (\overline{P} \cdot S \cdot \overline{D}) + R3 \cdot (P \cdot \overline{S} \cdot D) + R2 \cdot (\overline{P} \cdot \overline{S} \cdot D) + R1 \cdot (P \cdot \overline{S} \cdot \overline{D}) + R0 \cdot (\overline{P} \cdot \overline{S} \cdot \overline{D})$$

たとえば、R7=0、R6〜R0=0とすれば、R=P・S・Dとなりパターンレジスタと、CPUのデータ、VRAMのデータの論理積がVRAMに書き込まれることになる。

また、R7=1、R6=1、R5〜R0=0とすれば、$R = P \cdot S \cdot D + \overline{P} \cdot S \cdot D = (P + \overline{P}) \cdot S \cdot D = S \cdot D$となりCPUのデータとVRAMのデータの論理積が書き込まれることになって、結局パターンレジスタの内容は無視されることになる。

ここでは説明の簡略化のために、DをVRAMのデータ、Pをパターンレジスタとした。しかし、DがVRAMのデータになるのは、I/Oポート04A4Hのbit10のRead Sourceが0のときだけである。また、Pがパターンレジスタになるのは、I/Oポート04A2Hのbit14〜13がすべて0のときだけである。なお、D、Pの値が他のものになっても、ラスタオペレーションのコードと演算の関係は変わらない。

■I/Oポート04A6H

I/Oポート04A6Hは、フォアグラウンドカラーを設定するポートである。I/Oポート04A2Hのbit14 FGCを有効にするとパターンレジスタの代わりにここで設定されたフォアグラウンドカラーが使われる。詳細はFigure 5のとおり。

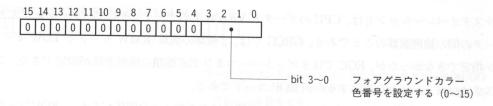



Figure 5　フォアグラウンドカラー

■I/O ポート 04A8H

I/O ポート 04A8H は、全プレーン共通のマスクレジスタである。詳細は Figure 6 とおり。

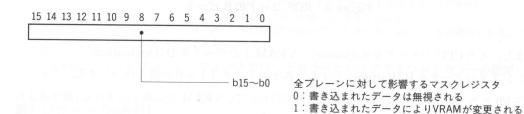



Figure 6　マスクレジスタ

■I/O ポート 04AAH

I/O ポート 04AAH は、バックグラウンドカラーを設定するポートである。I/O ポート 04A2H の bit13 の BGC を有効にするとパターンレジスタの代わりにここで設定されたバックグラウンドカラーが使われる。詳細は、Figure 7 のとおり。

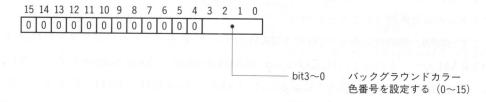



Figure 7　バックグラウンドカラー

■I/O ポート 04ACH

I/O ポート 04ACH は、EGC にあるブロック転送の設定ポートである。詳細は Figure 8 のとおり。

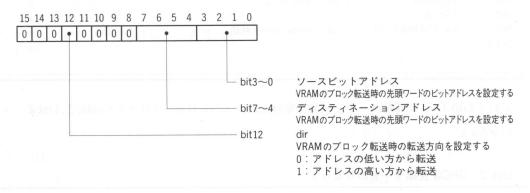



Figure 8　転送方向、転送アドレス

■I/O ポート 04AEH

I/O ポート 04AEH は、ブロック転送のビット長を設定するポートである。詳細は、Figure 9 のとおり。

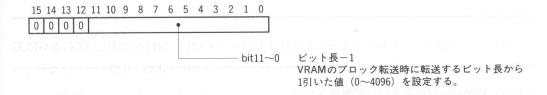



Figure 9　ブロック転送のビット長

GRCG 互換モード

GRCG 互換モードは、いままでの GRCG の機能とほぼ同等である。ただし、「GDC による 4 プレーン描画」の節で説明したように、いままでの GRCG では不可能な GDC によるアクセスが可能となっている。

モード変更は GRCG と同様に I/O ポート 7CH にデータを出力することにより行う。プログラムは List 1 のようになる。

List 1　GRCG 互換モードへの変更

```
push    0
pop     es
pushf
cli
mov     al,0C0h             ; GRCG RMW mode
out     7Ch,al
mov     es:[0495h],al       ; GRCG mode データ格納
popf
```

これでGRCGのRMWモードと同様な動作を行うようになる。リセットも同様で、List 2のようになる。

List 2　GRCG 互換モードのリセット

```
push    0
pop     es
pushf
cli
mov     al,00h              ; GRCG Reset
out     7Ch,al
mov     es:[0495h],al       ; GRCG mode データ格納
popf
```

ただし、この方法でモード切り替えを行う場合、I/Oポート6AHに04Hを出力し、EGCをGRCG互換モードにしておく必要がある。実際は、マシンのリセット時にGRCG互換モードになっているため、通常はこの設定を行う必要はない。

拡張モードのON/OFF

GRCGをONにしたあと、拡張モードに移行することにより、EGC本来の機能が使えるようになる。GRCGをONにするプログラムはList 3のようになる。

List 3　GRCG の ON

```
push    0
pop     es
pushf
cli
mov     al,0C0h            ; GRCG ON
out     7Ch,al
mov     es:[0495h],al      ; GRCG mode データ格納
popf
```

拡張モードに移行するためには、I/O ポート 6AH に 05H を出力すればよいが、このモード設定はモード変更フリップフロップが可能の状態になっていなければ変更できない。そこでまず I/O ポート 6AH に 07H を出力することによりモード変更フリップフロップを可能にし、モード設定が終了したらモード変更フリップフロップを変更できない状態に戻す (06H を出力する) ことになる。プログラムは List 4 のようになる。

List 4　拡張モードへの変更

```
mov     al,07h             ; モード変更 F/F 可能
out     06Ah,al
mov     al,05h             ; 拡張モード
out     06Ah,al
mov     al,06h             ; モード変更 F/F 不可
out     06Ah,al
```

これで拡張モードへ移行できるわけだが、このままでは EGC の多くのレジスタが不定のままである。そこで適切な初期値をすべてのレジスタに設定する必要がある。拡張モードの初期設定を行うプログラムは List 5 のようになる。

List 5　拡張モードの初期値設定

```
;---------- アクティブプレーンを全プレーンに設定
                mov     dx,04A0h
                mov     ax,0fff0h
                out     dx,ax
;---------- FGC,BGC = OFF , リードプレーンを全プレーンに設定
                mov     dx,04A2h
                mov     ax,0ffh
                out     dx,ax
```

```
;---------- マスクレジスタ = FFFFh
                mov     dx,04A8h
                mov     ax,0ffffh
                out     dx,ax
;---------- DIR = 0, ソース, デスティネーションビット = 0
                mov     dx,04ACh
                mov     ax,0
                out     dx,ax
;---------- ビット長 = 15
                mov     dx,04AEh
                mov     ax,15
                out     dx,ax
```

I/OポートレジスタPage 04A4Hのモードレジスタと ROP レジスタは、各グラフィック処理ルーチンで設定されるものなので、ここではセットしない。また、フォアグラウンドカラー及びバックグラウンドカラーは、それぞれI/Oポート04A2HのFGC、BGCがセットされていないと設定できないためここでは初期化しない。

拡張モードをOFFにするには、ONのときと同様にモード変更を可能とし、I/Oポート6AHに04Hを出力し、モード変更を不可にもどす。プログラムはList 6のようになる。

List 6　拡張モードのOFF

```
mov     al,07h          ; モード変更 F/F 可能
out     06Ah,al
mov     al,05h          ; GRCG 互換モード
out     06Ah,al
mov     al,06h          ; モード変更 F/F 不可
out     06Ah,al
```

そのあとGRCGをOFFにして、完全にEGCが動作しない状態となる（List 7参照）。

List 7　GRCGのOFF

```
pushf
cli
mov     al,0            ; GRCG mode off
out     7Ch,al
mov     es:[0495h],al   ; GRCG mode data set
popf
```

なお、拡張モードへ移行する際、EGCのレジスタの初期設定は省略できる場合があるが、安定した動作を目指すためには、できるだけ初期設定を行う方がよい。

4プレーンブロック移動

EGCのもう1つ大きな機能は、グラフィックVRAMの4プレーン同時ブロック移動である。これはハードウェアでビットシフト機能も実現している。

グラフィックを扱うソフトウェアでは、グラフィックVRAMの転送の処理がもっともCPUパワーを費やす。この処理を、このEGCの機能を使えば大幅に高速化できる。処理が複雑なので本書では解説を省いたが、添付のディスクにはこの機能を利用するためのソースコードが入っているので、興味のある方は参考にしてもらいたい。

EGCによる塗り潰し四角形の描画

EGCのラスタオペレーションは非常に強力な機能である。CPUデータとVRAMデータ及びパターンレジスタのデータとのありとあらゆる論理演算を行うことができ、その結果をVRAMに書き込むことができる。これは、グラフィックパターンの重ねあわせや、グラフィックキャラクタの背景上での移動などに威力を発揮する。

GRCG編で、GRCGにおけるRMWモードについて解説したが、これはVRAMへの書き込みの動作が理解しやすいものである。ここでは、EGCのGRCG互換モードではなく、拡張モードにおいてGRCGにおけるRMWモードと同じ動作を実現することにより、ラスタオペレーション（ROP）の設定を説明し、そのモードを使って塗り潰し四角形描画関数を作成する。ROPコードを変更すれば、GRCGでは不可能な描画ができる。

▶ポイント

- EGCの動作モードを拡張モードに設定する。
- 描画色の設定はパターンレジスタを使わずに、フォアグラウンドカラーを利用する。
- I/Oポート04A4Hのモードレジスタを設定し、描画するデータとソースデータ、パターンレジスタの設定方法を決定する。
- I/Oポート04A4HのROPコードをGRCGのRMWモードと同じになるように設定する。
- 実際の四角形の描画は「GRCGによる塗り潰し四角形の描画」と同じ方法で行う。
- 最後にEGCをOFFにする。

▶プログラミングテクニック

1 EGCを拡張モードにセットする

まずは、EGCの基礎知識で解説した方法でEGCを拡張モードにする。このときのEGCの各設定は、つぎのようになっている。

- ●アクティブプレーンは全プレーン
- ● FGC と BGC は無効、リードプレーンは全プレーン
- ●マスクレジスタ＝FFFFH でマスクは行わない
- ● DIR＝0、ソースとデストネーションのビット＝0
- ●ビット長＝16

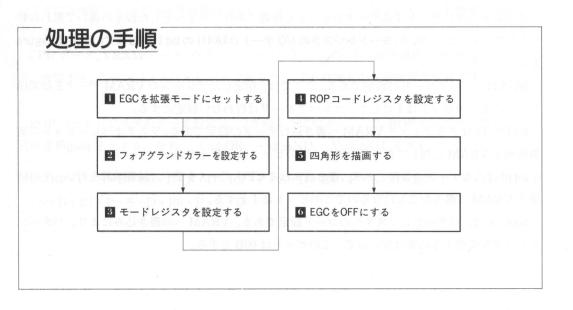
この時点でモードレジスタと ROP コードレジスタの設定は行われていない。

2 フォアグラウンドカラーを設定する

GRCG では、同じカラーコードのデータを描画するとき、プレーンに対応した 4 つのパターンレジスタに 00H または FFH をセットする必要があった。しかし、EGC の拡張モードでは、カラーコードを直接 I/O ポートに出力することにより、パターンレジスタの設定と同じことを行うことができる。この色の設定は 2 組あり、1 つがフォアグラウンドカラー、もう 1 つがバックグラウンドカラーである。

パターンレジスタか、フォアグラウンドカラーまたはバックグラウンドカラーを使うかの選択は I/O ポート 04A2H の bit14 と bit13 で行う（Table 1 参照）。

Table 1　I/O ポート 04A2H

bit14	bit13	選択されるパターンデータ
0	0	パターンレジスタ
1	0	フォアグラウンドカラー
0	1	バックグラウンドカラー

ここでは、フォアグラウンドカラーを使う。I/Oポートに出力してフォアグラウンドカラーを使うプログラムをList 1に示す。

List 1　フォアグラウンドカラー使用

```
mov     ax,040FFh       ; FGC ON
mov     dx,04A2h
out     dx,ax
```

パターンデータの選択が終わった時点でカラーコードの設定をI/Oポート04A6Hの下位4ビットに対して行う(バックグラウンドカラーではI/Oポート04AAH)。プログラムをList 2に示す。

List 2　フォアグラウンドカラーの設定

```
mov     ax,Color
mov     dx,04A6h        ; Foreground Color
out     dx,ax
```

ここで注意することは、カラーコードの設定は、選択されたパターンデータのみに対して行えるということである。たとえば、バックグラウンドカラーを選択して、フォアグラウンドカラーの設定を行おうとしてもそれは不可能である。

3 モードレジスタを設定する

このレジスタにセットするデータがもっとも複雑であり、ちょっとした設定の違いで思わぬ動作をしてしまうことになる。モードレジスタのI/Oポート04A4Hのbit13〜8までの設定は、Figure 1ようになる。

bit13はコンペアリードの設定である。ここでは、指定したプレーンのVRAMデータとは関係しないのでこのビットは1にして、コンペアリードは行わない設定にする。

bit12〜11はグラフィックVRAMへ書き込むデータの設定である。ラスタオペレーションの演算結果をVRAMに対して書き込むので、このビットは01Bとする。

bit10は入力データの設定である。書き込まれるCPUデータを使って論理演算を行い、その結果をVRAMに書きむことになるのでこのビットは1とする。

bit9〜8は、パターンレジスタへのロード設定である。VRAMへの書き込みにより、パターンレジスタを変更する必要はないので、このビットは00Bとする。

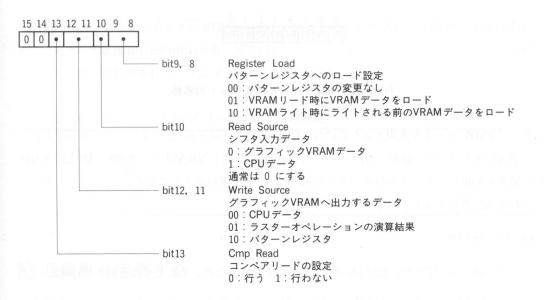



Figure 1　モードレジスタ（I/O ポート 04A4H）

以上から、モードレジスタに設定する値は101100Bとなる。この値をI/O ポート 04A4H の bit13〜8 として出力する。実際には、ROP コードを決定した後で、同時に出力する。

4 ROP コードレジスタを設定する

GRCG の RMW モードとは、CPU から書き込むデータと各タイルレジスタのデータの AND をとり、一方 CPU データの NOT をとったデータとすでにグラフィックプレーンにあるドットデータの AND をとり、両者の OR をとった結果をグラフィック VRAM に書き込むモードである。このように3種類のデータの論理演算は、EGC のラスタオペレーションのもっとも得意とするものである。

CPU データ、VRAM データ、パターンデータの3つのデータの論理演算を ROP コードレジスタで設定する。このレジスタは8ビットであるため、256種類もの論理演算を定義できることになる。

ROP コードレジスタのデータ形式は、つぎのようになる。ここで、ROP レジスタの各ビットの名称は Figure 2 とする。なお、＋は OR、・は AND、$\overline{X}$ は NOT を表す。

$$R = R7 \cdot P \cdot S \cdot D + R6 \cdot \overline{P} \cdot S \cdot D + R5 \cdot P \cdot S \cdot \overline{D} + R4 \cdot \overline{P} \cdot S \cdot \overline{D} + R3 \cdot P \cdot \overline{S} \cdot D + R2 \cdot \overline{P} \cdot \overline{S} \cdot D + R1 \cdot P \cdot \overline{S} \cdot \overline{D} + R0 \cdot \overline{P} \cdot \overline{S} \cdot \overline{D}$$

7	6	5	4	3	2	1	0
R7	R6	R5	R4	R3	R2	R1	R0

Figure 2　ROP コードの各ビットの名称

さて、RMW モードを実現するためには、CPU データ（以下 S）とパターンデータ（以下 P）との AND をとり、その結果と CPU データの NOT した値と VRAM データ(以下 D)との AND した結果を OR したデータを与えるような ROP コードを求めることになる。

これを式で表現すると、つぎのようになる。

$(S\cdot P)+(\overline{S}\cdot D)$

ここで、$(S+\overline{S})$，$(P+\overline{P})$，$(D+\overline{D})$の値はすべて 1 であるため、それをそれぞれの項に乗じる。

$(S\cdot P)\cdot(D+\overline{D})\ +\ (\overline{S}\cdot D)\cdot(P+\overline{P})$

この式を展開すると、

$S\cdot P\cdot D+S\cdot P\cdot \overline{D}+\overline{S}\cdot D\cdot P+\overline{S}\cdot D\cdot \overline{P}$

となる。これを並び替えると、つぎのようになる。

$P\cdot S\cdot D+P\cdot S\cdot \overline{D}+P\cdot \overline{S}\cdot D+\overline{P}\cdot \overline{S}\cdot D$

この式を実現する ROP コードレジスタは、

R7＝R5＝R3＝R2＝1
R6＝R4＝R1＝R0＝0

となる。したがって ROP コードレジスタの値は、Figure 3 のようになる。

7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	0

Figure 3　RMW モードを実現する ROP コードの各ビット

3 **4** の結果、モードレジスタとROPコードレジスタの値をあわせると、I/O ポート 04A4H に出力するデータは 0010110010101100B＝2CACH となる。この値を出力してモードレジスタと ROP コードレジスタの設定を行う。プログラムを List 3 に示す。

List 3　モードレジスタと ROP コードレジスタの設定

```
mov     dx,04A4h
mov     ax,02CACh
out     dx,ax
```

5 四角形を描画する

「GRCG を利用した塗り潰し四角形描画」と同様な方法で VRAM に対して描画を行う。

6 EGC を OFF にする

「EGC の基礎知識」で解説した方法で EGC を OFF にする。このとき各 EGC のレジスタに対する終了処理は省略しているが、すべてのレジスタを初期設定の値に戻しても良いだろう。

ライブラリ

EgcGraphicBoxf

解説　グラフィック画面に EGC を利用して塗り潰し四角形を描画する。書式はつぎのとおり。

void **EgcGraphicBoxf**(int *X1*,int *Y1*,int *X2*,int *Y2*,int *Color*)

X1,Y1　ボックス左上端座標
X2,Y2　ボックス右下端座標
Color　塗り潰す色番号

GRCG を利用した GraphicBoxf 関数とまったく同じ動作をする。
EGC タイプの機種のみで動作可。EGC 搭載をチェックする方法は「GDC による 4 プレーン描画」参照のこと。

戻り値　なし

EGCによるグラフィック画面への文字表示

「グラフィック画面への文字表示」で紹介したKanjiPutc関数は、GRCGのRMWモードを利用して描画を行っているため、グラフィック画面の文字の上に違う文字の上書きを行った場合、文字が重なって表示されてしまう。そこで、上書きを行う前に前の文字を消去しておく必要があった。

しかし、EGCのラスタオペレーションを適切に設定すれば、消去せずにグラフィック文字の上書きが実現できる。ここでは漢字のグラフィック画面への描画の章で紹介したKanjiPutc関数を、EGCのラスタオペレーションを使って実現する。

▶ポイント

- このプログラムはEGCを前提としてるので、文字のパターンデータはCGウィンドウから読み出す。
- カラーの設定はパターンレジスタを使わずに、フォアグラウンドカラーを使う。
- I/Oポート04A4Hのモードレジスタを「EGCによる塗り潰し四角形の描画」と同じに設定する。
- I/Oポート04A4HのROPコードを描画先のデータに依存しないように設定する。
- 「グラフィック画面への文字表示」と同じ方法で文字データの描画を行う。

▶プログラミングテクニック

1 CGウィンドウから漢字のパターンを読み出す

最初にKCG（漢字キャラクタジェネレータ）から漢字のパターンを読み出す。読み出しの方法については「グラフィック画面への文字表示」を参照のこと。なお、EGCタイプのPC-9801はすべてCGウィンドウを備えているため、ここではCGウィンドウを使って漢字パターンの読み出しを行う。

2 EGC の設定を行う

EGC を拡張モードに設定し、カラーの設定のためにパターンレジスタではなくフォアグラウンドカラーを設定し、I/O ポート 04A4H のモードレジスタを設定する。これらの設定は「EGC による塗り潰し四角形描画」とすべて同じである。

3 ROP コードレジスタを設定する

VRAM のデータを無視し、CPU データとパターンデータとの AND した結果を求める演算式を求める。これを式で表すとつぎのようになる。なお、＋を OR、・を AND、$\overline{X}$ を NOT とする。

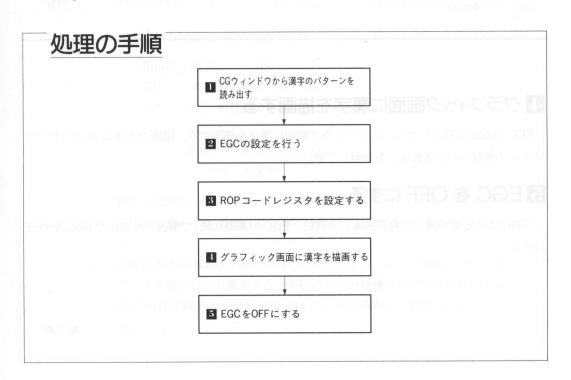
S・P

これに$(D+\overline{D})$を乗じる。

$(S・P)・(D+\overline{D})$

この式を展開すると

$S・P・D+S・P・\overline{D}$

となる。これを並び替えると、つぎのようになる。

$P \cdot S \cdot D + P \cdot S \cdot \overline{D}$

この式を実現するためのROPコードレジスタの値は、

R7＝R5＝1

R6＝R4＝R3＝R2＝R1＝R0＝0

となる。したがってROPコードレジスタの値は、Figure 1のようになる。

7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	0

Figure 1　文字描画で使うROPコードの各ビット

2 **3** の結果、モードレジスタとROPコードレジスタの値をあわせると、I/Oポート04A4Hに出力するデータは0010110010100000B＝2CA0Hとなる。この値を出力してモードレジスタとROPコードレジスタの設定を行う。プログラムをList 1に示す。

List 1　モードレジスタとROPコードレジスタの設定

```
mov     dx,04A4h
mov     ax,02CA0h
out     dx,ax
```

4 グラフィック画面に漢字を描画する

EGCの設定を終了したら、グラフィック画面に漢字を描画する。描画の方法については「グラフィック画面への文字表示」を参照して欲しい。

5 EGCをOFFにする

「EGCによる塗り潰し四角形描画」と同様、「EGCの基礎知識」で解説した方法でEGCをOFFにする。

ライブラリ

EgcKanjiGputc

解説　EGC を利用してグラフィック文字表示を行う関数。書式は以下のとおり。

void **EgcKanjiGputc**(unsigned int *Kanji*,int *XP*,int *YP* int *Color*)

Kanji　漢字シフト JIS コード
XP　表示 横座標 0〜639−16
YP　表示 縦座標 0〜400−16
Color　表示カラー
　bit8=0　普通文字
　　=1　太文字

EGC の拡張モードを利用して、全角文字をグラフィック画面の指定したドット位置に表示する。引数 COLOR の bit 8 を 1 にすると、太文字が描画される。
GRCG 利用の KanajiGputc 関数と異なり、グラフィック画面にすでにあるデータを消去して上書きする。EGC タイプの機種のみで動作可能である。
EGC の存在調査は「GDC による 4 プレーン描画」を参照のこと。

戻り値　なし

キーボード編

キーボードの基礎知識

PC-9800シリーズのキーボードには、機種によってさまざまなタイプがある。これらは、追加されたキーや特定の機種だけに存在するキーなど、タイプによっていくつかの違いがある。ここでは、キーボードの操作の具体的な解説の前に、現在までにどのようなキーボードが存在したかをまとめた。

キーボードの種類

PC-9800シリーズのキーボードには、Table 1に示すような6タイプがある。このタイプ分けは本書独自のものであるが、どのキーが備わっているのかということと、CAPSキーやカナキーのソフトウェアによるON/OFF制御が可能かどうかというプログラム設計時に考慮を要するような、機能的な差異にもとづいて分類している。

『テクニカルデータブック』では、初代、VM、XA、LTタイプのようにソフトウェアからCAPSキーやカナキーの制御ができないキーボードを旧キーボード、RA、LSタイプのようにそれができるキーボードを新キーボードと呼んでいる。新キーボードは、本体からCAPSキーやカナキーのロック制御をするためにコマンドを受信できるようになっている。ほかに新キーボードが接続されていることを確認するためのコマンドも公開されている。

キーボードインターフェイスコネクタの電気的な仕様は全PC-9800シリーズ（コネクタを持たない機種を除く）で同一であり、新キーボードと旧キーボードは互換性がある。したがって、本来新キーボードを持つ機種に旧キーボードを接続しても正しく動作するし、またその逆も可能である。ただし、後者の場合は電源OFFまたはリセットするとCAPSキーやカナキーロック状態は保持されない。また、BIOS経由ではvfキーの押下は認識できない。

初代タイプ

このタイプのキーボードを持つ機種はかなり初期のもので、NFERキーがないのはこのタイプだけである。NFERキーを利用するプログラムは、キーカスタマイズ可能にするか、入力できなくても支障のない機能を割り当てるように作成するほうが良いだろう。

Table 1　PC-9800 シリーズのキーボードの種類

	キーの有無				ロック制御の有無			
	NFER	HOME	vf・1〜5		HELP	CAPS・カナ	NUM	新旧
初代タイプ*	×	×	×	×	○	×	×	旧
VM タイプ*2	○	×	×	×	○	×	×	旧
XA タイプ*3	○	○	○	○	○	×	×	旧
LT タイプ*4	○	×	×	×	×	×	×	旧
RA タイプ*5	○	×	○	○	○	○	×	新
LS タイプ*6	○	×	○	○	○	○	○	新

＊1 初代のキーボードには [{]、[}]、[~]、[|] の刻印がないが、入力はできる。搭載機種は、PC-9801 初代/E/F/M。
＊2 キートップ上に ['] の刻印のある機種もあるが、入力はできない。搭載機種は、V30 機（ラップトップ,ノート含む）、PC-9801VX/UX/XL/XL²（ノーマルモード）/HA/DO。
＊3 [vf・1] 〜 [vf・5] の呼称は [f・11] 〜 [f・15]。XL、XL² のノーマルモードでは、[HOME]、[f・11] 〜 [f・15] が押されたときキーボードからのデータ送信は行われているがキーボード BIOS で読み出すことはできないので VM タイプに分類する。搭載機種は、PC-98XA、PC-9801XL/XL²（ハイレゾモード）。
＊4 キートップ上に ['] の刻印があるが、入力はできない。LT 上位互換の PC-98HA は VM タイプ。搭載機種は、PC-98LT。
＊5 本体から CAPS・カナのロック状態をソフトウェアで制御可能。搭載機種は、PC-9801RA 以降の 80286、80386、80486 機（ハイレゾモード含む）。
＊6 本体から CAPS・カナ・NUM のロック状態をソフトウェアで制御可能。DIP SW 2-7 が ON（キーボード識別ビット 0000:0481H bit 6,3 = 0,1）のとき、アプリケーションは [vf・1] 〜 [vf・5] が押されたら [f・6] 〜 [f・10] と同様の動作をするように求められている。搭載機種は、ラップトップ、ノート型の 80286、80386 機。
＊7 [HELP]、[BS]、[DEL]、[ROLL UP]、[ROLL DOWN]、[COPY]

VM タイプ

このタイプのキーボードを持つ機種は、RA タイプのキーボードを持つ機種と並んで多い。新キーボードが登場したあとも、V30 搭載機には新キーボードと同じデザインの VM タイプのキーボードが使用されている。

このキーボードには vf キーがない。vf キーを持たない機種は非常に多いので、これを利用するプログラムはキーカスタマイズ可能なように作成したほうがよい。なお、XL のノーマルモード時は、[HOME] キーと vf キーがキーボード BIOS から取得できないのでこのタイプに分類している。

VX 以降の機種には [^] キーに“‘”の刻印があるが、[SHIFT] を押しながら [^] キーを押しても“‘”は入力できない。過去の機種との互換性のためだと想像しているが、キートップに“‘”の刻印がある理由は不明だ。

XAタイプ

このタイプのキーボードを持つ機種はPC-98XA/XL/XL²のハイレゾモードだけである。PC-98RL以降のハイレゾ機にはRAタイプが装備されている。

このキーボードには[HOME]キーが単独存在するが、このキーを持つ機種は上記のとおり非常に限られているので使用しないほうがよいだろう。

LTタイプ

このタイプのキーボードを持つのはPC-98LTだけで、LT上位互換のPC-98HAはVMタイプのキーボードを備えている。

このキーボードには[HELP]、[BS]、[DEL]、[ROLL UP]、[ROLL DOWN]、[COPY]キーがない。また、[BS]キーと[DEL]キーのかわりに[DEL/BS]キーがある。[DEL/BS]キーは[BS]キー、[DEL]キーのどちらとも少し異なる動作をするので注意が必要である。

キーボードBIOSのファンクション00Hキーデータの読み出しを用いて[DEL/BS]キーを読み出すと、AH（キーコード）=39H、AL（キーデータ）=08Hが返る。Table 2にPC-98LT以外の機種の[BS]キー、[DEL]キー、[DEL/BS]キーのキーコード、キーデータの値を示したが、[DEL/BS]キーは、プログラムからは[DEL]と[BS]が混ざったように見えることがわかる。

個々のキーに割り当てられた番号であるキーコードが[DEL]と同じということは、[EL/BS]キーが押されると、キーボードユニットからは[DEL]キーと同じキーコードが発生するということを意味する。キーに割り当てられた文字コードであるキーデータが[BS]の値であるということは、[DEL]のキーコードを受け取ったキーボードBIOSのハンドラが、[BS]のキーデータに変換したということになる。

つまり[DEL/BS]キーは、キーボードBIOSでキーコードをチェックしたときやキーボードBIOSのファンクション04Hキー押下状態のセンスで押下状態をチェックしたときには[DEL]キーと同等に見え、キーボードBIOSでキーデータをチェックしたときには[BS]キーと同等に見えるということになる。

LTタイプでも[^]キーに〝'〟の刻印があるが、VMタイプと同様入力はできない。

Table 2　DEL、BS、DEL/BSのキーコード、キーデータの比較

	キーコード（AH）	キーデータ（AL）
[DEL/BS]	39H	08H
[DEL]	39H	00H
[BS]	0EH	08H

RAタイプ

RA以降のデスクトップ型80286、80386、80486搭載機にはノーマル・ハイレゾを問わずすべてこのキーボードが採用されている。このタイプは今後のPC-9800シリーズの標準的なキーボードであると言えるだろう。

このキーボードでは従来ハイレゾモードのみにあったvfキーがノーマルモードにも備わった。さらに大きな特徴は、マルチタスク対応のためソフトウェアから[CAPS]、[カナ]のロックのON/OFF制御ができるようになったことである。

LSタイプ

80286、80386を搭載したラップトップやノート型にはこのキーボードが採用されている。

このキーボードはRAタイプのキーボードとほぼ同等であるが、[CAPS]、[カナ]とともに[NUM]キーのON/OFF制御もソフトウェアから可能となっている。ところが、この3個のキーのON/OFF状態の設定は一度の書き込みで行うようになっている。[CAPS]、[カナ]はシステム共通域から現在の状態を知ることができるが、[NUM]の状態は知ることができないので、NUMの状態を保存したままCAPS・カナのON/OFF制御をすることができない。LSタイプのキーボードでは[CAPS]、[カナ]のON/OFF制御をしないか、NUMの状態が変化してもよいという前提で制御しなければならない。

なお、このタイプのキーボードではNUMロック状態のとき[f・6]〜[f・10]キーが[vf・1]〜[vf・5]に変化する。しかしNUMロック中に[f・6]〜[f・10]を入力したいような場合もあるだろう。それを想定して、このタイプのキーボードを持つ機種ではDIP SW 2-7でvfキーの用途をアプリケーションに通知するようになっている。アプリケーションは、DIP SW 2-7がONになっているときに[vf・1]〜[vf・5]が入力されたときには、[f・6]〜[f・10]と同様の動作をするように求められている。DIP SW 2-7の状態はシステム共通域のキーボード識別ビット（0000:0481H bit6、3）にも反映されている。しかし、このような処理は本来BIOS側でエミュレートを行うべきもので、アプリケーション側に対応を求めるのは間違いといえる。望ましい対応方法は、キー割り当てをカスタマイズできるようにアプリケーションを設計することである。

キーボードへのコマンド 送信と受信

新型キーボードは、CPU側からのコマンドデータを受け付けることができ、それに対してアクノリッジデータを返してくる。このあと紹介するキーボードタイプの取得や CAPS キーや カナ キーのコントロールなどでこの機能を利用するが、キーボードへのコマンド送信とアクノリッジ受信処理はハードウェアの構成と密接な関係があり、煩雑な処理が必要になる。そこで、ここではまずコマンドの送信とアクノリッジの受信について解説する。なお、後の節ではここで作成したルーチンを呼んで処理している。

▶ポイント

- 新キーボードは、コマンドを受け取るとアクノリッジを返す。コマンドを正常に受け取ったときにはFAH，コマンドの再送信を要求するときにはFCHが送られてくる。もし、これ以外の値が4回続けて送られてきたときや、一定時間待ってもアクノリッジが全く返ってこないときにはエラーと判断する。
- アクノリッジの受信データをキーボード割り込みハンドラに奪われないように、キーボード割り込みを割り込みコントローラで禁止してから実行する。この設定はこのルーチンを呼ぶ上位のプログラムで行わなければならない。なぜなら、キーボードタイプの識別のようにコマンド受け取りのアクノリッジを返したあとに、さらにデータを送信してくる場合があるからである。

▶プログラミングテクニック

1 8251Aの送信を許可する

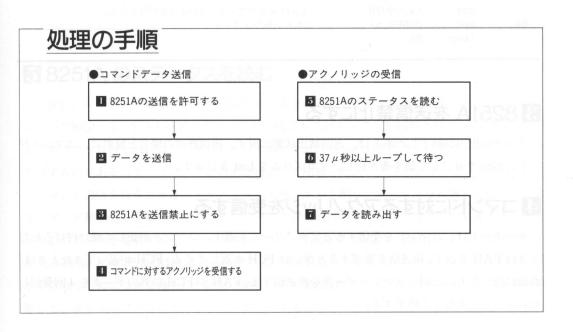
最初に、CPU側からキーボードにコマンドを送る方法を解説する。

本体側のキーボードインターフェイスに使われている8251Aは、通常キーボードからデータを受け取るだけなので送信禁止の状態になっているが、本ルーチンではキーボードにコマンドデータを送信するために、まず8251Aの送信を許可する。

送信を許可するには、8251AにI/Oポート43H（コマンドワード）のbit0を1にした値を書き

込む。コマンドワードのビットごとの意味はTable 1の通りである。OUT命令で書き込みが終わったら、I/Oのリカバリタイムとして4μ秒以上のウェイトを入れる。プログラムリストをList 1に示す。

Table 1 I/Oポート43Hの意味

	8251Aの動作	キーボードインターフェイスの動作
bit7	EH（同期モードのみ）	―
bit6	リセット (1=する、0=しない)	1=8251Aをリセットする 0=リセットしない
bit5	RTS端子 (1=LOW、0=HIGH)	1=RDY信号は常にインアクティブ 0=RDY信号はRxRDYの状態に従う
bit4	エラーフラグクリア (1=する、0=しない)	1=8251AのFE,OE,PEをクリアする 0=エラークリアしない
bit3	TxDATAブレーク送信 (1=する、0=しない)	1=RST端子をLOWレベルにする キーボードをリセットする 0=RST端子をHIGHレベルにする
bit2	受信許可 (1=する、0=しない)	1=キーボードからデータ受信する 0=キーボードからデータ受信しない
bit1	DTR端子 (1=LOW、0=HIGH)	1=RTY信号をHIGH（インアクティブ）にする 0=RTY信号をLOW（アクティブ）にする キーボードにデータの再送を要求する
bit0	送信許可 (1=する、0=しない)	1=RST端子からデータ送信する 0=RST端子からデータ送信しない

List 1　8251A の送信許可

```
SendRetry:      mov     al,00010111B    ;KB 用 8251A の設定
                        ;   |||||||+-KB I/F 送信設定 (1=許可)
                        ;   ||||||+--DTR#設定 (1=LOW) KB リトライ非要求
                        ;   |||||+---KB I/F 受信設定 (1=許可)
                        ;   ||||+----ブレーク送信設定 (0=しない) KB リセットしない
                        ;   |||+-----エラーフラグのクリア (1=する)
                        ;   ||+------RTS#設定 (0=HIGH) KB からの送信を許可
                        ;   |+-------リセット (0=しない)
                        ;   +--------Don't care
                out     0043H,al
                mov     cx,0007H        ;4 μ秒以上のウェイト (I/O リカバリタイム)
        @@:     out     005FH,al        ;(0.6 μ秒のウェイト)
                loop    @b
```

2 データを送信する

キーボードに送るコマンドデータを送信する。8251A の送信レジスタ（I/O ポート 41H）にデータを書き込めばキーボードにデータが送られる。この場合もリカバリタイムを 4μ 秒以上入れる。プログラムを List 2 に示す。

List 2　キーボードへのデータの送信

```
        mov     al,ah
        out     0041H,al
        mov     cx,0007H        ;4 μ秒以上のウェイト (I/O リカバリタイム)
@@:     out     005FH,al        ;(0.6 μ秒ウェイト)
        loop    @b
```

3 8251A を送信禁止にする

データの出力が終了したあとは、送信禁止状態に戻す。送信許可の場合と反対に、コマンドワードの bit0 を 0 にして値を書き込む。プログラムを List 3 に示す。

4 コマンドに対するアクノリッジを受信する

キーボードは、コマンドを受信するとアクノリッジを返す。コマンドが正しく受け付けられたときは FAH を返す。再送信を要求するときには FCH を返してくる。FCH が返ってきたときは、直前に送ったものと同じコマンドデータを再送信する。FAH や FCH 以外のデータを 4 回受信したらエラーとみなして終了する。

List 3　8251A を送信禁止にする

```
mov     al,00010110B    ;KB 用 8251A のモード設定
        ;  ^^^^^^^^
        ;  |||||||+-KB I/F 送信設定 (0=禁止)
        ;  ||||||+--DTR#設定 (1=LOW) KB リトライ非要求
        ;  |||||+---KB I/F 受信許可
        ;  ||||+----ブレーク送信設定 (0=しない) KB リセットしない
        ;  |||+-----エラーフラグのクリア (1=する)
        ;  ||+------RTS#設定 (0=HIGH) KB からの送信を許可
        ;  |+-------リセット (0=しない)
        ;  +--------Don't care
out     0043H,al
```

List 4　コマンドのアクノリッジ受信

```
            mov     bl,00H
AckRetry:   call    KbReceiveData
            jc      TimeOut
            cmp     al,0FAH         ;ACK ?
            jne     NotAck
            clc                     ; 正常終了
            ret
NotAck:     cmp     al,0FCH         ;NACK ?
            je      SendRetry
            inc     bl
            cmp     bl,04H
            jne     AckRetry        ; 非 ACK, 非 NACK を 4 回受信したらエラー終了
TimeOut:    stc
            ret
```

5 8251A のステータスを読む

つぎに、キーボードにコマンドを送信すると返されるアクノリッジを受信するためのルーチンについて解説する。キーボードへのコマンド送信でもこのルーチンが利用されている。また、キーボードコマンドのなかには、コマンドに対するアクノリッジのあとにさらにアクノリッジデータを返すものがあるので、コマンド送信とは独立して用意した。

コマンド送信を行うときは、アクノリッジをキーボード割り込みハンドラに奪われないようにキーボード割り込みを禁止しているので、データの受信はポーリングで行う。8251A がキーボードからのデータを受信するとステータスレジスタ（Table 2 参照）の RxRDY が 1 になるので、10 ミリ秒(以上)の間 RxRDY をチェックする。RxRDY=1 になったらキーボードからのデータを受け取るためにつぎに進む。十分長い間待っても RxRDY が 1 にならなかったときは異常と判断して、キャリーフラグをセットしてリターンする。

Table 2　ステータスレジスタ

	8251A の動作	キーボードインターフェイスの動作
bit7	DSR (1＝LOW, 0＝HIGH)	1＝KB コネクタ (6) が HIGH レベル 0＝KB コネクタ (6) が LOW レベル
bit6	同期検出 (同期モードのみ)	—
bit5	FE (フレーミングエラー) (1＝あり, 0＝なし)	1＝フレーミングエラーあり 0＝フレーミングエラーなし
bit4	OE (オーバーランエラー) (1＝あり, 0＝なし)	基本的に発生しないはず
bit3	PE (パリティエラー) (1＝あり, 0＝なし)	1＝パリティエラーあり 0＝パリティエラーなし
bit2	TxEMP	1＝全ての送信データの送信が終わっている
bit1	RxRDY	1＝キーボードからデータを受信した
bit0	TxRDY	1＝送信データの書き込みが可能

List 5　ステータスの読み出し

```
                mov     cx,8000H        ;タイムアウトするまでのRDY#チェック回数
RdyCheckLoop:   in      al,0043H        ;RDY#チェック
                test    al,02H
                jnz     Ready
                loop    RdyCheckLoop
                stc                     ;タイムアウトなのでキャリーをセット
                ret
Ready:
```

6 37μ 秒以上ループして待つ

RxRDY＝1 になってから 37μ 秒経過後データを読み出す。$\overline{\text{RDY}}$ を 37μ 秒以上インアクティブ状態に保つためである。このウェイトを置かないと、キーボードは本体がデータを引き取ったことを認識できないため、つぎの処理に進まなくなってしまう。

List 6　37μ 秒のループ

```
                mov     cx,3EH          ;37μ秒以上ウェイト
ReceiveLoop:    out     005FH,al
                loop    ReceiveLoop
```

7 データを読み出す

エラーが検出されなければ、$\overline{\text{RTY}}$をインアクティブにして受信データを読み出し、キャリーフラグをクリアしてリターンする。エラーが検出された場合は、$\overline{\text{RTY}}$をアクティブにして受信データを読み出し、キャリーフラグをセットしてリターンする。

List 7　データの読み出し

```
        ;ステータス読み出し
                in      al,0043H
                and     al,38H
        ;エラーがあればRTY#=ACTIVE、なければRTY#=INACTIVE
                jnz     ReceiveError
                mov     al,00010110B        ;RTY#=1(INACTIVE)
                out     0043H,al
                mov     cx,0007H            ;4μ秒以上のウェイト(I/Oリカバリ)
        @@:     out     005FH,al            ;(0.6μ秒ウェイト)
                loop    @b
                in      al,0041H            ;受信データ読み出し
                clc                         ;キャリーをクリア
                ret
ReceiveError:   mov     al,00010100B        ;RTY#=0(ACTIVE)
                out     0043H,al
                mov     cx,0007H            ;4μ秒以上のウェイト(I/Oリカバリ)
        @@:     out     005FH,al            ;(0.6μ秒ウェイト)
                loop    @b
                in      al,0041H            ;受信データ読み出し
                stc                         ;キャリーをセット
                ret
```

ライブラリ

KbSendCommand

解説　キーボードへのデータ送信。このプログラムはアセンブラから呼び出すサブルーチンである。C言語から直接利用することはできない。AHレジスタにキーボードに送信するコマンドデータを入れて呼び出す。このプログラムの実行の前後はキーボード割り込みを禁止する。割り込みの禁止と解除は、このルーチンを呼ぶプログラムの責任で行わなければならない。

戻り値　コマンドの実行結果。キャリーフラグに結果が返される。

キャリーフラグの値	状　態
0	正常終了
1	エラー終了

サンプル　CAPSキー、カナキーのコントロールをする関数・キーボードを取得する関数から呼び出されている。

KbRecieveData

解説　キーボードからのデータ受信を行う。このプログラムはアセンブラから呼び出すサブルーチンである。C言語から直接利用することはできない。このプログラムの実行の前後はキーボード割り込みを禁止する。割り込みの禁止と解除は、このルーチンを呼ぶプログラムの責任で行わなければならない。

戻り値　受信したデータと、実行結果を返す。キャリーフラグに実行結果、ALレジスタにキーボードから受信したデータが入る。

キャリーフラグの値	状　態
0	正常受信（AL：キーボードからの値）
1	タイムアウトまたはエラー

サンプル　CAPSキー、カナキーのコントロールをする関数・キーボード種別を取得する関数から呼び出されている。

COLUMN

PC-9800シリーズ発売時期順年表

Table AはPC-9800シリーズの機種名を発売時期の順番に並べたものである。

PC-9801には、ある世代から標準で装備されるようなった機能や、逆にある世代以降搭載されなくなった機能、またある世代から変更された機能がある。主要な相違点は『テクニカルデータブック』に明記されているが、微妙な部分の差異は明らかにされていない。たとえば、MS-DOS 3.1で拡張フォーマットが可能な内蔵型ハードディスクはPC-9801VX以降であるとか、カセットインターフェイスボードはPC-9801UV11以降の機種でサポートされなくなったなどの情報である。

そのようなものは、転換点がわかれば発売時期からおよその見当をつけることができる。ただし、PC-9801VM21のように特異な機種(CPUにV30を搭載しているにもかかわらずITF ROMやCPUリセットポート00F0Hを持っている、拡張バスの信号がPC-9801VX 2 と同様であるなど) もあるので、完全な判断材料になるわけではない。

Table A　PC-9800シリーズ発売時期順年表

発売時期	機種名
'82-10	PC-9801
'83-10	PC-9801F2
'83-11	PC-9801E
'84-10	PC-9801F3
'84-11	PC-9801M2
'85-02	FC-9801
'85-02	PC-9801M3
'85-05	PC-98XA model 1、2、3
'85-06	PC-9801U2
'85-07	PC-9801VM0、2
'85-07	PC-9801VF2
'85-10	PC-9801VM4
'86-03	FC-9801V
'86-05	PC-98XA model 11、21、31
'86-06	PC-9801UV2
'86-10	PC-9801VX0、2、4
'86-10	PC98-LT model 1、2
'86-11	PC-9801VM21
'86-12	PC-9801VX4WN
'86-12	PC-98XL model 1、2、4
'87-06	PC-9801VX01、21、41
'87-06	PC-9801UV21
'87-08	PC-9801VX41、WN
'87-09	FC-9801X model 1、2
'87-10	PC-9801UX21、41
'87-10	PC-98XL^{2}
'87-10	PC-98LT model 11、21
'88-03	PC-9801UV11
'88-03	PC-9801LV21
'88-04	PC-9801CV21
'88-08	PC-9801RA2、5
'88-09	PC-9801RX2、4
'88-10	FC-9801V2
'88-11	PC-9801LS2.5
'88-11	PC-9801VM11
'88-11	PC-98LT model 22
'89-02	PC-98RL model 2、5
'89-02	PC-9801LV22
'89-02	FC-9801X model 21
'89-04	PC-9801ES2、5
'89-04	PC-9801EX2、4
'89-04	PC-9801LX2、4、5
'89-06	PC-98DO
'89-07	FC-9801A
'89-08	PC-9801LX5C
'89-10	PC-9801RX21、51
'89-11	PC-9801N
'89-11	PC-9801RA21、51
'89-11	PC-9801RS21、51
'90-02	PC-H98 model 70
'90-02	PC-9801T model W2、W5
'90-06	PC-9801NS
'90-06	PC-9801T model F5
'90-07	PC-9801T model S5
'90-09	PC-H98 model 100、U100
'90-09	PC-H98 model 60、U60
'90-09	PC-98RL model 21、51
'90-10	PC-98DO+
'90-10	FC-H98 model 100
'90-11	PC-9801DX2、5、U2、U5
'90-11	PC-9801NV
'90-11	PC-98HA
'91-01	PC-9801DA2、5、7、U2、U5、U7
'91-01	PC-9801DS2、5、U2、U5
'91-01	PC-9801UR、UF
'91-03	PC-9801NV(Blue、Pink)
'91-03	RC-9801
'91-05	PC-9801NS/E
'91-05	PC-H98S model 8
'91-05	FC-9801U
'91-07	PC-9801T mode F51、F71、W7
'91-10	PC-9801CS
'91-10	PC-9801NC
'91-10	PC-98GS
'91-11	PC-H98 model 90、U90
'91-11	PC-H98 model 80、U80
'91-11	SV-H98 model 30
'92-01	PC-9801FA2、5、7、U2、U5、U7
'92-01	PC-9801NS/T
'92-01	PC-9801NL

CAPS キー・カナキーの制御

新キーボードのCAPSキーやカナキーは、従来の機械式のロックキーからソフトウェアで制御するロックキーに変わった。それにともない、本体からキーボードにコマンドを送ることでCAPSやカナキーのロックのON/OFF状態を制御できるようになった。ここでは前節で解説したキーボードへのコマンドの送信を使って、ソフトウェアでCAPSキーやカナキーを制御する方法を解説する。

▶ポイント

- コマンドを送信するとキーボードからはアクノリッジが返される。これをキーボード割り込みハンドラに奪われないようにキーボード割り込みを禁止しておく。
- ロック制御コマンドでは、CAPSとカナのロック状態を同時に設定しなければならない。一方の状態だけを変化させたいときには、変化させないキーの状態をシステム共通域から得て、変化させるキーの状態と合わせてロック状態を送信する。
- コマンドを送ってCAPSやカナのロック状態を変更しても、本体のキーボードハンドラにはそのことが通知されない。実際のロック状態とキーボードハンドラ内部の状態を一致させるために、BIOSでキーボードの初期化を行う。

▶プログラミングテクニック

1 キーボード割り込みを禁止する

キーボードにコマンドを送るときには、前もってキーボード割り込みを禁止しておく必要がある。キーボードにコマンドを送出するとアクノリッジデータが返ってくるが、それをキーボード割り込みハンドラに横取りされるのを防ぐためである。アクノリッジデータはコマンドを送出したルーチンで受け取って処理をする。

キーボード割り込みは、キーボードインターフェイスの8251がキーボードからシリアルデータbitを受信すると発生する。キーボード割り込みはマスタ割り込みコントローラ(マスタPIC)の入力1に接続されているので、割り込みを禁止するにはマスタPICの割り込みマスクレジスタのbit 1を1にする(Table 1参照)。

Table1　キーボード割り込みのマスク用のI/Oポート

I/Oポート	意　味
02H	割り込みコントローラ（マスタ）の割り込みマスクレジスタ bit1　意　味 0　キーボード割り込み許可 1　キーボード割り込み禁止

プログラムはList1のようになる。

List1　キーボード割り込みの禁止

```
cli                     ; INとOUTの間に割り込みが入ってIMRが変更
                        ; されないように割り込み禁止
in      al,0002H        ; マスタ割り込みコントローラのIMR読み込み
or      al,02H          ; bit 1 = 1
out     0002H,al        ; マスタ割り込コントローラのIMR書き込み
sti
```

2 ロック制御コマンドを送出する

ロック制御コマンドは2バイトで構成されていて、最初に9DHを、つぎにCAPSとカナのロック状態を通知するための値を送る（Table 2参照）。キーボードからはアクノリッジをい示す2つのコードが返ってくる（Table 3参照）。コマンド送信に関係する処理は別の節で解説したコマンド送信ルーチンにまかせるので、このルーチンでは正常終了かエラー終了かだけを判断する。コマンド送信ルーチンは、一定時間内にキーボードから正常終了のアクノリッジがこないとエラーを返す。

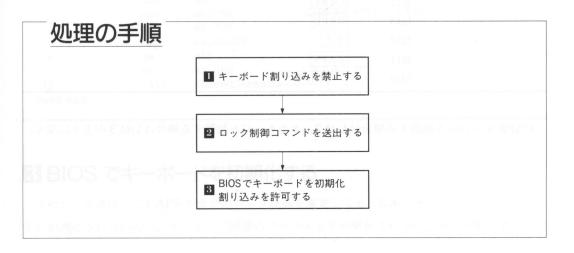



Table 2　ロック制御コマンド

送信順序	意　味	内　容	
第1バイト	キーロック制御	9DH	
第2バイト	制御内容のパラメータ	bit0	NUM ロック（1＝ON、0＝OFF）
		bit1	0
		bit2	CAPS ロック（1＝ON、0＝OFF）
		bit3	カナ ロック（1＝ON、0＝OFF）
		bit4～bit6	1
		bit7	0

Table 3　キーボードからのアクノリッジ

意　味	値
コマンド再送要求アクノリッジ	FCH
コマンド正常終了アクノリッジ	FAH

ロック制御コマンドの2バイト目を見ればわかると思うが、CAPSキーとカナキーの状態はかならず一緒にキーボードに送らなければならない。どちらかの状態だけを変化させたいときには、変化させないキーの状態をシステム共通域から得て、変化させるキーの状態と合わせてロック状態を送信する。CAPSキーとカナキーの現在の状態は**システム共通域0000：053AH**を見ればわかる（Table 4参照）。

Table4　CAPSキーとカナキーの押下状態を示すシステム共通域

アドレス	意　味	
0000H:053AH	シフトキーの押下状態フラグ（KB_SHFT_STS）	
	ビット	意味（1＝押されている、0＝押されていない）
	bit4	CTRL
	bit3	GRPH
	bit2	カナ
	bit1	CAPS
	bit0	SHIFT

CAPSキーのみを制御する場合はList 2、カナキーのみ制御する場合はList 3のようになる。

List 2　CAPSロック制御コマンド送出

```
        ;ロック制御コマンド送出
                mov     ah,9DH                  ;LED制御コマンド
                call    KbSendCommand
                jc      send_end                ;送信終了
        ;ON/OFF情報送出
                push    es
                mov     ax,0000H                ;ES←システム共通域
                mov     es,ax
                mov     ah,es:[053AH]
                pop     es
                shl     ah,1
                and     ah,08H                  ;カナ制御ビットを残しマスク
                or      ah,70H
                cmp     sw,0000H
                je      off
                or      ah,04H                  ;CAPS ON
off:            call    KbSendCommand
send_end:
```

List 3　カナロック制御コマンド送出

```
        ;ロック制御コマンド送出
                mov     ah,9DH
                call    KbSendCommand
                jc      send_end                ;送信終了
        ;ON/OFF情報送出
                push    es
                mov     ax,0000H                ;ES←システム共通域
                mov     es,ax
                mov     ah,es:[053AH]
                pop     es
                shl     ah,1
                and     ah,04H                  ;CAPSキー制御ビットを残しマスク
                or      ah,70H
                cmp     sw,0000H
                je      off
                or      ah,08H                  ;カナキーON
off:            call    KbSendCommand
send_end:
```

3 BIOSでキーボードを初期化する

コマンドを送信してCAPSやカナのロック状態を変更しても、本体のキーボードハンドラにはそれが通知されないので、キーボード関連のワークエリアが更新されない。その状態でキー入力

するとロック状態変更前の文字が入力されてしまうことになる。自前のプログラムでワークエリアを設定すればよいのだが、かなり複雑な作業が必要である。そこで、BIOSのキーボード初期化コマンドを実行してBIOSに設定させることにする。キーボードの初期化を実行するとBIOSがキーボードにリセット信号を送る。すると、キーボードは現在押されている(ロックされている)キーの情報を通知してくる。キーボードハンドラはその情報を受け取ると内部のワークエリアを更新する。これで実際のキーのロック状態とキーボードハンドラのワークエリアの状態が一致する。

なお、キーボードを初期化したあとは、必ずキーボード割り込みを許可をしなければならない。

プログラムはList4のようになる。

List4　BIOSによるキーボードの初期化

```
quit:    ; キーボードBIOS初期化
                mov     ah,03H
                int     18H
         ; キーボード割込許可
                cli
                in      al,0002H
                and     al,NOT 02H
                out     0002H,al
                sti
```

▶ワンポイント

■実行タイミング

キーボードリセット後、0〜3ミリ秒と10〜50ミリ秒の期間はキーボードにコマンドを発行してはならないことになっている(3〜10ミリ秒の期間は良い)。動作を確実にするためには、このルーチンの最初に50ミリ秒のウェイトを入れれば良いのだが、そうすると常に非常に長い実行時間がかかることになってあまり好ましくない。

そこで、このルーチンにはウェイトを入れず、これを呼ぶプログラムの責任で前記の期間にこのルーチンを実行しないように注意することにする。本ルーチンの最後ではキーボードリセットを行っているので、本ルーチンを連続して実行する場合にも同様の注意が必要になる。

■NUM キーのコントロール

LS タイプのキーボードは RA タイプのキーボードとほぼ同等であるが、CAPS や カナ と同様に NUM キーの ON/OFF 制御もソフトウェアから可能となっている。ところが、この 3 個のキーの ON/OFF 状態の設定は一度の書き込みで行うようになっている。システム共通域に、NUM キーの状態は示されていない。つまり NUM の状態を保存したまま CAPS や カナ の ON/OFF 制御をすることができない。LS タイプのキーボードでは CAPS や カナ の ON/OFF 制御をしないか、NUM の状態が変化しても良いという前提で制御しなければならない。

ライブラリ

CapsSwitch

解説

CAPS キーを ON/OFF する。書式は以下のとおり。

void **CapsSwitch** (unsigned int *sw*);

sw　ON/OFF スイッチ

数値	状態
0	OFF
1	ON

NUM ロックはかならず解除される

戻り値　なし

サンプル　CAPSSW.C

KanaSwitch

解説

カナキーを ON/OFF する。書式は以下のとおり。

void **KanaSwitch** (unsigned int *sw*);

sw　ON/OFF スイッチ

数値	状態
0	OFF
1	ON

NUM ロックはかならず解除される

戻り値　なし

サンプル　KANASW.C

キーボードタイプの判別

PC-9800 シリーズのキーボードを機能的な差異にもとづいて分類すると基礎知識で解説したように6つのタイプに分けることができる。すべての機種が備えていないキーやキーボードの機能を使用するプログラムは、使用可能な機種を制限するか、機種に応じた処理を行う必要がある。

ここでは、システム共通域の情報からキーボードタイプを判別する方法を解説する。なお、別節でキーボードへコマンドを送ってキーボードタイプの判別を行っているが、システム共通域を参照する方がより細かなキーの分類が可能である。

▶ポイント

- 新キーボードをもつ機種の場合、システム共通域のキーボード識別ビットからキーボードの種類を判別できる。
- 旧キーボードをもつ機種の場合、機種を調べてキーボードの種類を判別する。機種はシステム共通域の機種情報ビットから判断できる。
- 機種情報ビットは PC-98LT と PC-98HA で同じ値を示しているのでさらに分類が必要になるのだが、システム共通域の情報から確実に PC-98LT と PC-98HA を区別する方法は明らかになっていない。そこで、PC-98HA にのみ存在するメモリカード BIOS を実行して、その結果から PC-98LT と PC-98HA の判別を行う。

▶プログラミングテクニック

1 新キーボードを判別する

新キーボードであれば、システム共通域のキーボード識別ビット（Table 1 参照）だけで種類を判断できる。旧キーボードの場合は 2 以降で処理をする。

キーボードのタイプを判別したら、判断結果としてキーと機能の有無（基礎知識の Table 1 の分類と同じ）を bit 6〜0 に割り当てた値を返す。各ビットの割り当ては Table 2 のとおりである。また、プログラムを List 1 に示す。

Table 1　システム共通域のキーボード識別ビット

アドレス	意　味		
0000:0481H	キーボードタイプ（KEYB_TYPE）		
	bit6	bit3	キーボードの種類
	1	1	新キーボード・LSタイプ（vf・1～vf・5→f・6～f・10変換不要）
	0	1	新キーボード・LSタイプ（vf・1～vf・5→f・6～f・10変換必要）
	1	0	新キーボード・RAタイプ
	0	0	旧キーボード

Table 2　判別結果の格納形式

ビット	機　能
bit 6	vf・1～vf・5→f・6～f・10変換（1=必要、0=不要）
bit 5	NFER（1=あり、0=なし）
bit 4	HOME（1=あり、0=なし）
bit 3	vf・1～vf・5、バッククォート（1=あり、0=なし）
bit 2	BS、DEL、HELP、ROLL UP、ROLL DOWN（1=あり、0=なし）
bit 1	CAPS、カナ ソフトウェア制御機能（1=あり、0=なし）
bit 0	NUM ソフトウェア制御機能（1=あり、0=なし）

List 1　新キーボードの判別

```
;---つぎに示すように、AH にキーボード情報を入れる
;bit 6 = 0000:0481 bit 6 (キーボード識別 1)
;bit 3 = 0000:0481 bit 3 (キーボード識別 2)
        push    es
        mov     ax,0000H        ; システム共通域
        mov     es,ax
        mov     ah,es:[0481H]   ; キーボード識別 1,2 → AL bit 6,3
        and     ah,48H
    ;--- タイプ分類
        cmp     ah,40H          ;RA タイプ判別
        mov     al,0101110B     ;      /NFER/    /vf/HELP/LOCK/
        je      Quit
        cmp     ah,48H          ;LS タイプ判別 (vf 変換不要)
        mov     al,0101111B     ;      /NFER/    /vf/HELP/LOCK/NUM
        je      Quit
        cmp     ah,08H          ;LS タイプ判別 (vf 変換必要)
        mov     al,1100111B     ; 変換/NFER/    /  /HELP/LOCK/NUM
        je      Quit
```

2 旧キーボードを判別する

キーボード識別ビットのbit6とbit3がともに0なら旧キーボードである。この場合はシステム共通域の機種情報ビット(Table 3参照)から機種を調べてキーボードの種類を判別する(Table 4参照)。ここではPC-9801初代、XA、VMの3つのタイプを判別して、PC-98LT/HAの処理は3で行う。

キーボードのタイプを判別するプログラムをList 2に示す。

Table 3　システム共通域の機種情報ビットとキーボード識別

アドレス	意味	
0000:0500H	BIOS 制御用フラグ 0 (BIOS_FLAG)	
	bit0	機種情報 1 (PC-9801 初代/E/F/M、PC-98LT/HA 判別)
	1	その他
	0	PC-9801 初代/E/F/M、 PC-98LT/HA、 ハイレゾモード
0000:0501H	BIOS 制御用フラグ 1 (BIOS_FLAG)	
	bit4	機種情報 3 (PC-9801U2、PC-98LT/HA 判別)
	1	PC-9801U2、PC-98LT/HA
	0	その他
	bit3	機種情報 4 (ハイレゾ、ノーマル判別)
	1	ハイレゾモード
	0	ノーマルモード

Table 4　機種情報の値と機種の対応

機種情報 1	機種情報 3	機種情報 4	判定結果
0	0	0	PC–9801 初代/E/F/M
×	×	1	ハイレゾモード
0	1	0	PC–98LT/HA
1	×	0	上記以外

List 2　旧キーボードの判別

```
;--- 旧キーボード判別
        ;--- つぎに示すように、AH に機種情報を入れる
        bit 7 = 0000:0500 bit 0 ( 機 種 情 報1)
        bit 6 = 0000:0501 bit 4 ( 機 種 情 報3)
        bit 5 = 0000:0501 bit 3 ( 機 種 情 報4)
                mov     ah,es:[0500H]   ; 機種情報 1 → bit7
                and     ah,01H
                ror     ah,1
                mov     al,es:[0501H]   ; 機種情報 3, 機種情報 4 → bit 6,5
                and     al,18H
                shl     al,1
                shl     al,1
                or      ah,al
        ;--- タイプ分類
                cmp     ah,00H          ; 無印タイプ判別
                mov     al,0000100B     ;     /    /    /  /HELP/    /
                je      Quit
                test    ah,20H          ;XA タイプ判別
                mov     al,0111100B     ;     /NFER/HOME/vf/HELP/    /
                jnz     Quit
                test    ah,80H          ;VM タイプ判別
                mov     al,0100100B     ;     /NFER/    /  /HELP/    /
                jnz     Quit
```

3 PC-98LTとPC-98HAを判別する

システム共通域の情報からPC-98LTとPC-98HAを確実に区別する方法は明らかになっていない。そこで、PC-98HAにのみ存在するメモリカードBIOSを実行して、その結果からPC-98LTとPC-98HAの判別をする（Table 5参照）。

PC-98LTはメモリカードBIOSをサポートしていないので、PC-98LTで実行するとINT 1FH実行前のAXの値（B600H）が返される。PC-98HAで実行した場合にはAHにステータス情報が入るためAXにB600Hが返されることはない。

Table 5　PC-98HAのメモリカードBIOS

ファンクション	内　容		
INT 1FH B6H	カードサイズの取得		
		入力	BX＝00H
		出力	CF＝終了条件（異常終了時は1、正常終了時は0） AH＝ステータス情報 BX＝メモリカードサイズ PC−98LTで実行するとAX＝B600Hで終了する PC−98HAではAX＝B600Hにはならない

キーボードのタイプを判別したら、判断結果としてキーと機能の有無をTable 2と同じ形式で返す。プログラムをList 3に示す。

List 3　PC-98LTとPC-98HAの判別

```
;--- PC-98LT/HA 判別
                push    bx
                mov     ax,0B600H           ; カードサイズ取得コマンド
                int     1FH                 ; メモリカード BIOS
                pop     bx
                cmp     ax,0B600H           ;LT なら AX の値は不変, HA ならかならず変化
                mov     al,0100000B         ;     /NFER/    /  /     /     /   (LT)
                je      Quit
                mov     al,0100100B         ;     /NFER/    /  /HELP/      /   (HA)
Quit:           mov     ah,00H
                pop     es
                ret
```

▶ワンポイント

■本体と異なるキーボードの組み合わせ

本体付属以外のキーボードを接続したときには、Table 6 のように判定される。

Table 6　付属キーボード以外と組み合わせたときの判別結果

本　体	判定結果
初代タイプ	初代タイプ
VM タイプ	VM タイプ
XA タイプ	XA タイプ
RA タイプ	VM タイプ（ノーマルモード） XA タイプ（ハイレゾモード）

ライブラリ

GetKeyType

解説　キーボードの種類を取得する。書式はつぎのとおり。

unsigned int **GetKeyType**(void)

戻り値　キーボードの機能有無を納めた値。各ビットの意味はつぎのとおり。

ビット	意　味
bit 6	vf・1〜5 → f・6〜10 変換（1＝必要、0＝不要）
bit 5	NFER（1＝あり、0＝なし）
bit 4	HOME（1＝あり、0＝なし）
bit 3	vf・1〜5、バッククォート（1＝あり、0＝なし）
bit 2	BS、DEL、HELP、ROLL UP、ROLL DOWN（1＝あり、0＝なし）
bit 1	CAPS,カナソフトウェア制御機能（1＝あり、0＝なし）
bit 0	NUM ソフトウェア制御機能（1＝あり、0＝なし）

サンプル　KEYTYPE.C

キーボードタイプ取得コマンドの実行

『テクニカルデータブック』では、新キーボードへのコマンドとしてCAPSキーとカナキーのロック制御コマンドとともにキーボードタイプ取得コマンドが公開されている。

別節で解説されているように、キーボードタイプはシステム共通域からも知ることができる。しかし、本来旧キーボードを持つ機種に新キーボードを接続したときにもそれを正しく認識できるような独自のキーボードハンドラを作りたいときには、キーボードに直接コマンドを送信してその種類を取得する必要がある。ここでは、公開コマンドのインプリメント例をかねて、キーボードタイプ取得コマンドを実行するプログラムを解説する。

▶ ポイント

- コマンドを送信するとキーボードからはアクノリッジが返される。これをキーボード割り込みハンドラに奪われないように、キーボード割り込みを禁止しておく。
- 新キーボードの場合、キーボードタイプ取得コマンドを送るとキーボードは2バイトのタイプ通知アクノリッジを返す。その値をチェックして新キーボードの判定を行う。

▶ プログラミングテクニック

1 キーボード割り込みを禁止する

キーボードにコマンドを送るときには、前もってキーボード割り込みを禁止しておく必要がある。キーボード割り込みハンドラにアクノリッジを横取りされるのを防ぐためである。具体的な割り込みの禁止の方法は、別節の「CAPSキー・カナキーの制御」と同じでマスタ割り込みコントローラの割り込みマスクレジスタのbit1を1にする(Table 1参照)。プログラムはList 1のようになる。

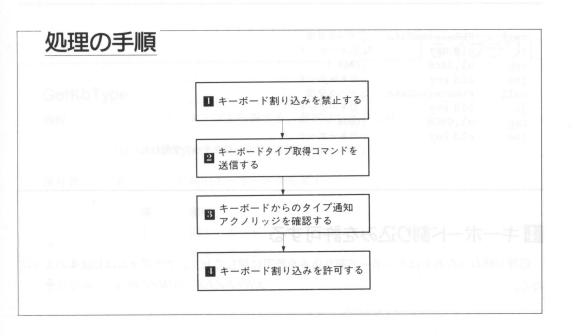
Table 1　キーボード割り込みの禁止

I/Oポート	意　味	
02H	割り込みコントローラ（マスタ）の割り込みマスクレジスタ	
	bit1	意　味
	0	キーボード割り込み許可
	1	キーボード割り込み禁止

List 1　キーボード割り込みの禁止

```
in      al,0002H       ; マスタ割り込みコントローラの IMR 読み込み
or      al,02H         ;bit 1 = 1
out     0002H,al       ; マスタ割り込みコントローラの IMR 書き込み
sti
```

❷ キーボードタイプ取得コマンドを送信する

キーボードに送る**キーボードタイプ取得コマンドは 9FH** である。キーボードにコマンドを送ると最初にコマンドに対するアクノリッジが返される。これでコマンドが正常に処理されたかどうかが判断できる。コマンド送信に関係する処理は別節で解説したコマンド送信ルーチンに任せる。このルーチンは、アクノリッジを見て正常終了かエラー終了かを返すので、それで判断する。また、コマンド送信ルーチンは、一定時間内にキーボードから正常終了のアクノリッジがこないとエラーで終了する。プログラムは List 2 のようになる。エラーで終了すれば、旧型キーボードである。

List 2　キーボードタイプ取得コマンドの発行

```
mov     ah,9FH          ; キーボードタイプ取得コマンド発行
call    KbSendCommand
jc      old_key         ; 正常に実行できなければ旧キーボード
```

3 キーボードからのタイプ通知アクノリッジを確認する

新キーボードは、キーボードタイプ取得コマンドが正常に実行されると、続いてA0H、80Hの順にタイプ通知アクノリッジを送信してくる(Table 2参照)。データ受信ルーチンを呼び出して、このとおりのデータを受信したことが確認できたら新キーボードと判断する。

プログラムはList 3のようになる。

Table 2　キーボードから送られてくるデータ

意　味	値
コマンド再送要求アクノリッジ	FCH
コマンド正常終了アクノリッジ	FAH
タイプ通知アクノリッジ第1バイト	A0H
タイプ通知アクノリッジ第2バイト	80H

List 3　タイプ通知アクノリッジのチェック

```
call    KbReceiveData   ; データ受信
jc      old_key         ; 旧キーボード
cmp     al,0A0H         ;0A0H ?
jne     old_key         ; 旧キーボード
call    KbReceiveData   ; データ受信
jc      old_key         ; 旧キーボード
cmp     al,080H         ;080H ?
jne     old_key         ; 旧キーボード
                        ; 新キーボードのアクノリッジをすべて受信した
```

4 キーボード割り込みを許可する

処理が終わったあとはキーボード割り込みを許可に戻しておく。プログラムはList 4のようになる。

List 4　キーボード割り込みの許可

```
in      al,0002H
and     al,NOT 02H
out     0002H,al
```

▶ワンポイント

■旧キーボードの場合

旧キーボードに対してコマンドを送出すると、キーボードリセットがかかってしまう。これは従来キーボードリセット用の信号線であるRST信号線でコマンドを送出するためである。

ライブラリ

GetKbType

解説　キーボードのタイプを取得する。書式はつぎのとおり。

int **GetKbType**(void)

戻り値　新キーボードか旧キーボードかを返す

値	意味
0	旧キーボード
1	新キーボード

サンプル　CAPSSW.C、KANASW.C

キーバッファのビープ機能の制御

キーボードBIOSは、キーバッファがオーバーフローしたときビープを鳴らす機能を持っている。この機能はシステム共通域を操作すれば制御できる。MS-DOS起動時はこの機能がアクティブになっているが、アクションゲームのようにキーボードを文字入力ではなくプッシュボタンのように使うようなアプリケーションでは、実行中は停止しておくほうがよい場合がある。もちろん、キーバッファに溜まっているデータを、定期的に読み捨てる方法もあるが、ビープ機能を止めてバッファをオーバーフローさせたままにしておくほうが簡単である。

ここでは、キーバッファオーバーフロー時のビープ機能の制御の方法とそれに関連する話題を取り上げる。

▶ポイント

- キーボードBIOSのキーボード割り込みハンドラは、キーバッファがオーバーフローするとビープを鳴らす。この機能はシステム共通域0000：0500H bit 5の操作で制御できる。

▶プログラミングテクニック

1 バッファオーバーフロー時のビープ機能の状態を取得する

キーボードBIOSのキーバッファがいっぱいになるとキーを押すたびにビープが鳴る。ハイレゾモード以外では、ブザーが鳴っている間はプログラムまで停止してしまう。これを避けるには、**システム共通域のキーバッファオーバーフロー通知ビット0000：0500H bit5**を操作してビープ機能を停止させる（Table 1参照）。

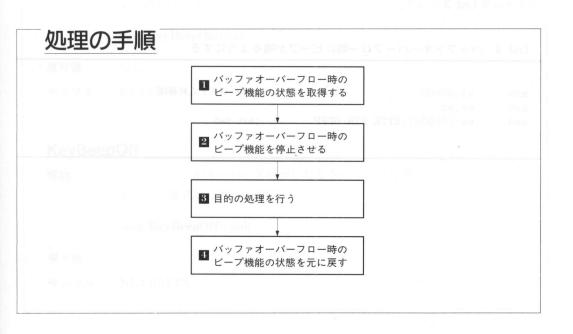
Table 1　キーバッファオーバーフロー通知ビット

アドレス	意　味	
0000:0500H	BIOS 制御用フラグ（BIOS_FLAG）	
	bit5	キーバッファオーバーフロー通知制御ビット
	1	オーバーフロー時にビープを鳴らさない
	0	オーバーフロー時にビープを鳴らす

アプリケーションの実行中にビープ機能を停止した場合、終了時には元の状態に戻しておくほうが望ましい。かならず鳴らすように設定しても実用上問題はないと考えられるが、最初に現在の状態を取得して、最後に元の状態に戻すようにするのがマナーにかなっている。

システム共通域 0000：0500H bit 5 の状態を取得するプログラムを List 1 に示す。

List 1　ビープ機能の状態の取得

```
mov     ax,0000H                ES ←システム共通域
mov     es,ax
mov     al,es:[0500H]         ;bit 5
and     ax,0020H
shl     al,1
shl     al,1
rol     al,1
```

2 バッファオーバーフロー時のビープ機能を停止させる

バッファオーバーフロー時のビープ機能を停止させるには、システム共通域の0000：0500H bit 5を1にする。プログラムをList 2に示す。

List 2　バッファオーバーフロー時のビープ機能を停止させる

```
mov     ax,0000H
mov     es,ax
or      es:[0500H],BYTE PTR 20H
```

3 目的の処理を行う

ビープ機能が停止している間に、目的の処理を行う。ここでは、とくに具体的な処理は説明しない。ゲームなど各自のプログラムの処理を行えばよい。

4 バッファオーバーフロー時のビープ機能の状態を元に戻す

アプリケーションを終了するときには、ビープ機能を元の状態に戻しておく。最初に取得した状態にしたがって、再開または停止するルーチンを呼び出す。OFFにするプログラムは先に示したので、ここではシステム共通域の0000：0500H bit 5を0にして、ビープが鳴るようにするプログラムをList 3に示す。

List 3　バッファオーバーフロー時にビープが鳴るようにする

```
mov     ax,0000H                        ;ES←システム共通域
mov     es,ax
and     es:[0500H],BYTE PTR 0DFH        ;bit 5=0
```

ライブラリ

GetKeyBeepMode

解説 キーバッファがオーバーフローしたときにビープを鳴らすモードかどうかを取得する。書式はつぎのとおり。

int **GetKeyBeepMode**(void)

戻り値 現在の状態。値と状態の対応はつぎのとおり。

値	状 態
0	ビープを鳴らす(デフォルト)
1	ビープを鳴らさない

サンプル KEYBEEP.C

KeyBeepOn

解説 キーバッファがオーバーフローしたときにビープを鳴らすモードに設定する。書式はつぎのとおり。

void **KeyBeepOn**(void)

戻り値 なし

サンプル KEYBEEP.C

KeyBeepOff

解説 キーバッファがオーバーフローしたときにビープを鳴らさないモードに設定する。書式はつぎのとおり。

void **KeyBeepOff**(void)

戻り値 なし

サンプル KEYBEEP.C

キー押下状態の読み取り

アクションゲームのようなアプリケーションでは、キーを押し続けるという操作がよく用いられる。こうしたプログラムのキー入力ルーチンには、MS-DOSのキー入力ファンクションやキーボードBIOS(INT 18H)の文字入力ファンクション(00H、01H、05H)を利用するより、キーボードBIOSのキー入力状態のセンス（04H）でリアルタイムなキーの押下状態を調べるほうが適している。

ただし、ほとんどのキーは0.5秒以上押しつづけるとリピートが始まるので、単純に押下状態を調べるとキーが離されているように見えてしまうことがある。この現象が起きるとプログラムの操作性が非常に悪くなるので、回避策が必要である。

そこでここでは、キーのオートリピートの影響がでない、キーの正しい押下状態を読みとる方法を解説する。

▶ ポイント

- キーを押し続ける操作をするアプリケーションのキー入力ルーチンでは、入力された文字を読みとる代わりにキーそのものの押下状態を調べる。
- キーリピートしている状態では、40ミリ秒周期で1ミリ秒の期間、キーが離されているように見える瞬間がある。これによる押下状態の誤認を避けるには、1ミリ秒強の間隔を置いて2回キーの状態を調べ、両方の結果をORをとればよい。人間のキー操作で1ミリ秒だけキーを離すことは不可能なので、キーリピートとキー操作によるキーOFFを見誤ることはない。
- キーボードBIOSから文字を読まないままキーを押しつづけると、キーバッファがあふれてビープが鳴るので、この関数を利用するアプリケーションはキーバッファオーバーフロー時のビープ機能をOFFにしておく。

▶プログラミングテクニック

1 キー押下状態を取得する

キーボード BIOS は先行入力を可能にするためにキーバッファを持っている。キー入力があると、キーボードはどのキーが押されたかという情報を本体に通知する。同様にキーが離されたときもその情報を通知する。本体側では、キーボードから情報を受けとると CPU にキーボード割り込みが発生する。キーボード割り込みハンドラは、キーボードからの情報を読み取り、キーが押されたときにはキーコードをキーバッファに追加する。一方、アプリケーションが**キーボード BIOS (INT 18H) の文字入力ファンクション (00H、01H、05H)** を実行すると、BIOS はキーバッファから文字を取り出してアプリケーションに引き渡す。キーボード BIOS はこのように先行入力のための処理をしているので、アプリケーションが文字入力以外の処理をしているときにキーを押しても、正しくキー入力ができるのである。通常、文字を入力するときには、このファンクションを利用するとよい。

さて、キーボード BIOS には、バッファの状態とは無関係に、現在どのキーが押されているのかを調べるファンクションも用意されている。キーボード割り込みハンドラは、キーが押された、離されたという情報を受けとるたびに、**システム共通域の 0000：052A〜053AH** にあるキーの押下状態テーブルを書き換える。アプリケーションがキーボード BIOS の**キー入力状態のセンス (04H)** を実行すると、BIOS はこのテーブルからキーの状態を取り出してアプリケーションに引き渡す (Table 1 参照)。キーの押下状態をリアルタイムに調べたいときにはこちらのファンクションを利用することになる。

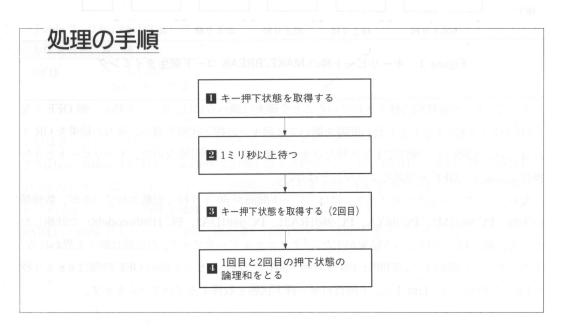



Table 1　キー押下状態のセンス

ファンクション	内　容
INT 18H 04H	キー押下状態のセンス 入力　AL＝キーコードグループ番号（00H～0FH） 出力　AH＝キーコードグループ内の8つのキーの状態、グループ内で押されているキーに対応するビットが1になる。 破壊　AX

ところが、ほとんどのキーは0.5秒(PC-9801初代は1秒)以上押しつづけると自動的にリピートしはじめる。リピートしないキーは、[f・1]～[f・10]、[vf・1]～[vf・5]、[CTRL]、[CAPS]、[SHIFT]、[カナ]、[GRPH]、[INS]のみである。このリピート処理はキーボード自身が行っていて、MAKE(ON)コードとBREAK（OFF）コードを連続的に発生させる。MAKE/BREAKコードの発生タイミングを実際に調べてみると、40ミリ秒周期のリピートのなかで1ミリ秒の間、キーが押されていないように見える瞬間が発生する(Figure 1参照)。ちょうどこのタイミングでキーの押下状態をチェックしてしまうと、キーが押され続けているにもかかわらずプログラムは押されていないと判断してしまうことになる。その確率は40ミリ秒のうちの1ミリ秒なので、約40回に1回と決して低くない率である。

Figure 1　キーリピート時のMAKE/BREAKコード発生タイミング

そこで、キーが連続的に押下されていることを確実に調べるには、リピート時に一瞬OFFとなる時間(1ミリ秒)よりも少し長い間隔を置いて2回キーの押下状態を調べ、両方の結果をORすればよい。人間のキー操作で1ミリ秒だけキーを離すことは不可能なので、キーリピートとキー操作によるキーOFFを見誤るようなことはない。

なお、『テクニカルデータブック』にはリピート間隔が60ミリ秒と記載されているが、数種類の実機（PC-9801M2、PC-98XA、PC-9801RA21、PC-9801DA2、PC-H98Smodel8）で計測したところ、40～42ミリ秒という結果が出た。『テクニカルデータブック』の記述は誤りと思われる。また、リピート開始までの時間は499ミリ秒～503ミリ秒、リピート時のOFF時間は0.8ミリ秒～1.0ミリ秒だった。List 1に、1回目のキー押下状態を取得するプログラムを示す。

List 1　キー押下状態の取得（1 回目）

```
mov     ax,KeyCodeGroup
mov     ah,04H
int     18H
mov     bl,ah
```

❷1 ミリ秒以上待つ

前述のとおり、2 回目のキー押下状態取得の前に 1 ミリ秒以上のウェイトを入れる。ただし、この関数の実行時間をなるべく短くするために必要以上に長くしないほうがよい。

PC-9800 シリーズには資源を占有せずに 1 ミリ秒程度の時間を計測する方法がとくに用意されていない。ここではおおよその時間ウェイトすればよいので CPU をループさせることにする。しかし、単純なループにすると高速な機種にあわせてループ回数を決定しなければならないため、低速な機種で実行したときに必要以上にウェイト時間が長くなってしまって実用上問題がある。CPU の種類とクロックからループ回数を調節することもできるが、かなり面倒である。

そこで、OUT 命令をはさんでループするようにした。ここで使用する **I/O ポート 5FH** は PC-H98 シリーズで用意されたウェイトポートで、0.6μ 秒のウェイトが保証される（Table 2 参照）。PC-H98 シリーズ以外ではこのポートは未使用なので OUT してもなんら問題はなく、かつ PC-H98 シリーズのようにほぼ一定時間のウェイトが入ることが判明している。これは、外部バスのタイミングの互換性を維持するため、ハードウェアが I/O ポートへの出力時にウェイトを入れるためだろう。

Table 2　0.6μ 秒（以上）ウェイト

I/O ポート	意　味
5FH	ウェイト用ポート bit7～0　任意

List 2 に、1 ミリ秒以上ウェイトするためのプログラムを示した。実行時間を実測してみると、20MHz の 80486（PC-H98S model8 で計測）のとき約 2.2 ミリ秒、8MHz の 8086（PC-9801M2 で計測）のとき約 3.7 ミリ秒だった。この程度の余裕があれば、OUT 命令の最低実行時間は保証されているので、さらに高速な機種で実行しても 1 ミリ秒以上のウェイトは得られるだろう。また、8MHz の 8086 でもウェイト時間に倍の違いはないので、目的を果すことができる。

List 2　1 ミリ秒以上のウェイト

```
            mov     cx,400H
waitloop:   out     005FH,al
            loop    waitloop
```

3 キー押下状態を取得する（2 回目）

1 回目と同じようにキーボード押下状態を取得する。プログラムを List 3 に示す。

List 3　キー押下状態の取得（2 回目）

```
mov     ax,KeyCodeGroup
mov     ah,04H
int     18H
```

4 1 回目と 2 回目の押下状態の論理和をとる

2 回の調査の結果を OR をとる。これにより、どちらかの回でビットが 1 になっていたら、結果は押下されていたことになる。プログラムを List 4 に示す。

List 4　1 回目と 2 回目の押下状態の論理和

```
or      ah,bl
mov     al,ah
ret
```

▶ワンポイント

■バッファオーバーフロー時のビープ機能に注意する

キーを押しつづけると、キーバッファにキーコードが蓄積される。普通はキーボード BIOS の文字入力ファンクションでキーバッファからキーコードを取り出すのだが、この関数では押下状態を調べるだけなので、キーバッファにはキーコードが溜まる一方である。キーバッファには 16 個のキーコードを蓄積できるが、キーバッファがいっぱいのとき、さらにキーが押されると、キーボード割り込みハンドラが警告のためにビープを鳴らす。ビープが鳴るとうるさいだけでなく、ハイレゾモード以外ではプログラムの実行まで停止してしまうので、KeyTouch 関数を利用する

ときは、この機能をOFFにしておくとよい。OFFの方法は「キーバッファのビープ機能の制御」の節を参照してもらいたい。

■キーバッファの問題

ビープ機能をOFFしておけば前述の問題は解決するが、キーバッファにはキーコードが溜まったままになっている。その状態でアプリケーションを終了させたりすると、MS-DOSのプロンプトに戻ったときに無意味な文字列が入力されてしまうようなことになって好ましくない。そのため、アプリケーション終了前にはキーボードBIOSの文字入力ファンクションでキーバッファに溜まっている文字をすべて読み捨てるようにする。

アプリケーションによっては、リアルタイムにキーボードの押下を調べたいときと、キーボードから文字を入力したいときとが混在することもあるだろう。そのときも、文字入力をする場面に入る前にキーバッファの内容を読み捨てればよい。

■実行時間

このプログラムを実行すると、中間のウェイトのために約2〜4ミリ秒程度の時間がかかる。アクションゲームのように複雑な処理をしながらキー入力も頻繁に行うアプリケーションではこの時間はばかにならない。そのようなときには、この関数をそのまま使わず、1ミリ秒以上の時間がかかる処理をはさんで、この関数のようなキー調査を行うように設計すればよいだろう。

ライブラリ

KeyTouch

解説　連続押下されているキーの押下調査。書式はつぎのとおり。

int **KeyTouch**(int *KeyCodeGroup*)

KeyCodeGroup BIOS AH=04 INT 18H のキーコードグループ番号

オートリピートするキーを押しつづけると、オートリピートのためにソフトウェアからはキーが押されていないように見えてしまう瞬間がある。この関数ではキーリピートしていても押しているかどうかを調べることができる。

戻り値　キーコードグループ内のキーの押下状態

サンプル　KEYTOUCH.C

図 キー押下状態を取得する（2回目）

```
mov   ax,KeyCodeGroup
mov   ah,04H
int   18H
```

図 1回目と2回目の押下状態の論理和をとる

```
or    al,cl
```

▶ワンポイント

シリアルポート編

シリアルポートの基礎知識

PC-9800シリーズは全機種にシリアルポート（RS-232Cシリアルインターフェイス）が標準装備されている。シリアルポートに関してはハイレゾモードはノーマルモードと互換性がある。PC-H98シリーズやPC-98HAなどはコネクタの形状が標準的な25ピンのD-SUBコネクタと異なるが、ソフトウェア的には他のPC-9800シリーズと同じと考えてよい。

PC-9800シリーズのシリアルポートには8251A（または同等品）という周辺LSIが使用されているが、実際にシリアルポートを直接操作する場合は他の周辺LSIの制御も必要になる。以降、順をおってシリアルポートを直接操作するためのノウハウを解説する。ここでは予備知識として、シリアルポート周辺のハードウェアの構成を解説する。

BIOSを利用しない理由

シリアルポートは主にモデムなどの比較的低速な装置と接続するために使用される。モデム以外にもプリンタ、マウス、デジタイザ、イメージスキャナ、FAXアダプタ、簡易LANユニットなどの機器を接続するときにも使用される。また、異機種間での通信手段として使われることも多い。これはシリアルポートにRS-232Cという汎用のシリアルインターフェイスを採用しており、多くのパソコンに標準で装備されているからである。

しかし、シリアルポートのBIOSによるサポートは大部分のパソコンで貧弱である。たとえば、PC-9800シリーズのRS-232CのBIOSには次のような制約がある。

- 19200bpsや38400bpsが設定できない。
- RS/CSフロー制御（ハードウェアフロー制御）がサポートされていない。
- 送信割り込みがサポートされていない。
- 通信中に通信パラメータを変更できない。
- ブレーク信号の送信をサポートしていない。
- 受信バッファの使用効率が悪い。

このように、RS-232CのBIOSが貧弱なのはPC-9800シリーズに限ったことではない。IBM-PC、J-3100、FMRなど、他の機種でもRS-232C関係のBIOSが貧弱で制約が多い傾向は同じである。このように制約の多いBIOSを使用していては実用的な通信ソフトを作成することはできない。

そこで多くの通信ソフトでは、BIOSを使用せずにシリアルポートのハードウェアを直接操作してこの制約を回避している。そこで、シリアルポートを直接操作し、実用になる通信ソフトを作れる能力を持つライブラリを作成する。ここで作成するライブラリはつぎのような能力を持つようにする。

- システムクロックが5MHz系ならば19200bpsや38400bpsを設定できる。
- XON/XOFFフロー制御およびRS/CSフロー制御をサポートしている。
- 送信割り込みをサポートしている。
- 通信中に通信パラメータを変更できる。
- ブレーク信号の送信をサポートしている。
- 受信バッファの使用効率が良い。

シリアルポートを使用した通信では、なんらかの方法で通信相手に転送を開始させたり停止させたりする必要があり、これをフロー制御と呼んでいる。

フロー制御にはコントロールコードの11H(XON)および13H(XOFF)を使用するXON/XOFFフロー制御(ソフトウェアフロー制御)と、RS-232Cの制御信号線のRS信号およびCS信号を使用するRS/CSフロー制御（ハードウェアフロー制御）がある。

シリアルポートに関わるハードウェア

シリアルポートの回路図の詳細は『テクニカルデータブック』に掲載されている。シリアルポートに関する機種間の相違は外部クロック入力用の切り替え回路くらいで、通常の調歩同期方式で使用する場合はソフトウェア的には相違はないと考えてよい。

Figure 1にPC-9800シリーズのシリアルポート全体の構成を示した。最近の機種では相当する回路がカスタムチップ内に集積されているが、ソフトウェア的には同じと考えてよい。シリアルポートを直接操作するということは、この中の8251、8253、8259、8255という4つの周辺LSIを操作するということである。通信を行うためには、この4つの周辺LSIを適切に初期化したりパラメータを設定したりする必要がある。

8251はシリアルインターフェイス専用のLSIで、シリアルポートの中心的な機能を担当している。実際のデータの送受信は、この8251で行われる。8251へは、通信パラメータとして、キャラクタ長、パリティの有無、ストップビット長などを設定する。

8253はプログラマブルタイマカウンタである。8253は3個のカウンタを内蔵しており、その内の1つが通信速度を設定するために使用されている。残りの2つは、ビープ音の音源や汎用のタイマなどに使用されている。

8259は割り込みコントローラで、シリアルポートのハードウェア割り込みを使用する場合には8259への設定も必要になる。

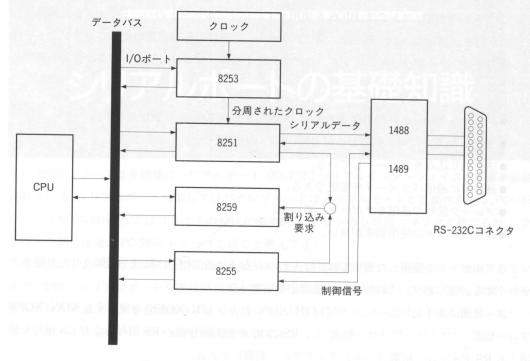



Figure 1　シリアルポートのハードウェア構成

8255はパラレルインターフェイス用のLSIで、PC-9800シリーズではシステムポートに使用されている。システムポートにはRS-232Cの制御信号線の状態を読み取ったり、シリアルポートの割り込み信号を選択したりするビットが用意されている。

1488と1489はパソコン内部のロジック信号とRS-232Cの信号の電圧レベルを変換するためのインターフェイス素子でありソフトウェアを作るうえでは関係ない。

これらの周辺LSIの機能の詳細は周辺LSIのデータシートなどを参照していただきたい。

周辺LSIとI/Oポート

PC-9800シリーズでは、シリアルポートに関連した周辺LSIはTable 1のようなアドレスのI/Oポートに割り当てられている。したがって、これらの周辺LSIの制御はアセンブラのIN命令やOUT命令などによりI/Oポートとやりとりして行うことになる。

これらの周辺LSIはもともと低速な8ビットCPU用に開発されたLSIであるため、16ビットあるいは32ビットの高速CPUが連続的にアクセスすると周辺LSI側の速度が追い付かないことがある。したがって、これらの周辺LSIに対してIN命令やOUT命令を続けてアクセスする場合は、それぞれのLSIに固有なリカバリタイムと呼ばれる時間間隔をおく必要がある。

各LSIにアクセスする場合のリカバリタイムはJMP ＄+2命令を使用するのが普通である。JMP ＄+2命令の必要数は、『テクニカルデータブック』を参照していただきたい。

Table 1　シリアルポート関連のI/Oアドレス

I/Oポート	Read/Write	意　味
00H	Write	割り込みコントローラ(8259)のコマンドレジスタ(ICW1)
02H	Read	割り込みコントローラ(8259)の割り込みマスクレジスタ(IMR)
	Write	割り込みコントローラ(8259)の割り込みマスクレジスタ(OCW1)
30H	Read	8251の受信データレジスタ
	Write	8251の送信データレジスタ
32H	Read	8251のステータスレジスタ
	Write	8251のモードレジスタ／コマンドレジスタ
33H	Read	システムポート(8255のポートB)
35H	Read/Write	システムポート(8255のポートC)
37H	Write	システムポート(8255のモードレジスタ)
75H	Read/Write	タイマ(8253)のデータレジスタ
77H	Write	タイマ(8253)のコマンドレジスタ

シリアルポートの初期化

シリアルポートを使用する場合は、通信速度、キャラクタ長、パリティ、ストップビットなどを決め、それらのパラメータを周辺LSIに適切に設定しなければならない。

ここでは、シリアルポートの初期化として各通信パラメータの設定の方法を解説する。なお、ここでは初期化を題材としているが、通信中の設定変更も基本的な手順も初期化と同じである。また、初期化として一度に設定しているが、各設定は独立しているものもあり、それらは個々に変更できる。独立できるものは関数を独立させてあるので、参考にしてもらいたい。

▶ポイント

- シリアルポートの初期化は8251に対する設定処理が主となる。
- 通信速度はプログラマブルタイマカウンタの8253に、キャラクタ長やパリティなどはシリアルインターフェイスの8251に設定する。
- 送受信処理をハードウェア割り込みを使用して行うので、システムポートの8255や割り込みコントローラの8259への設定も必要となる。
- 各LSIへの設定値のなかには読み出せないものもあるので、あとで参照するために設定値をメモリに保存する。

▶プログラミングテクニック

1 シリアルポートからの割り込みを禁止する

シリアルポートの初期化中は誤動作を防止するため、シリアルポートからの割り込みを禁止しておく必要がある。

シリアルポートの割り込みは、システムポート（I/Oポート35H）に設定する値によって、シリアルポートの割り込みに関して、〝発生しない〟、〝受信時のみ〟、〝送信時のみ〟、〝送受信時〟の4つが選択できる。

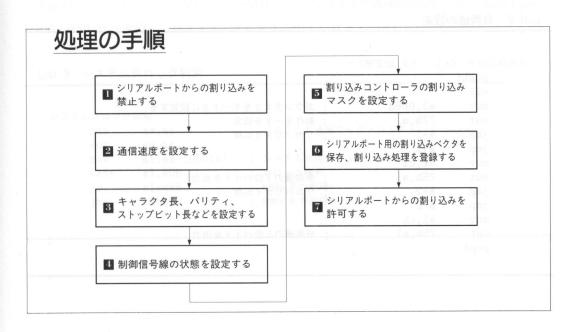
シリアルポートからの割り込みを禁止したい場合は、このうち〝発生しない〟に設定すればよい。具体的には、システムポートのbit0〜bit3を0にする。

システムポートの設定を変更する場合は、I/O ポート 35H から設定値を読み出して、変更したいビットだけを変更してから再び I/O ポート 35H に出力する(Table 1 参照)。シリアルポート関係以外のビットを変化させないように注意する必要がある。

Table 1　シリアルポートからの割り込みの禁止

I/O ポート	意味			
35H	8255 のポート C			
	ビット	信号	値	意味
	bit 0	RXRE	0	受信割り込みの禁止
			1	受信割り込みの許可
	bit 2	TXRE	0	送信割り込みの禁止
			1	送信割り込みの許可

なお、I/O ポート 37H を使用して、I/O ポート 35H をビット単位で変更することも可能である。ただし、I/O ポート 37H では 1 回の出力で 1 ビットしか変更できないので、複数のビットを変更する場合は I/O ポート 35H を使用するほうがよい。

送信割り込みを使用している場合は割り込み処理中でシステムポートの値が変わることがあるので、システムポートを読み出して値を変えて書き込む場合は、割り込み禁止状態で行う必要がある。プログラムを List 1 に示す。

List 1　RS-232C の割り込みを禁止する

```
; RS-232C の割り込みを禁止する
        pushf
        cli                             ; 割り込み禁止
        in      al,35h                  ; システムポートを読む
        jmp     $+2                     ; リカバリタイム用
        and     al,11111000b
        out     35h,al                  ; RS-232C の割り込みを禁止
        popf
```

2 通信速度を設定する

通信速度はプログラマブルタイマカウンタの 8253 に設定する。8253 は 3 個のカウンタ（カウンタ#0、カウンタ#1、カウンタ#2）を内蔵しており、PC-9800 シリーズではカウンタ#2 をシリアルポートの通信速度の設定用に使っている。

8253 へは I/O ポート 77H で動作モードを設定し、I/O ポート 75H で分周値を設定する。動作モードはモード 3（方形波レートジェネレータ）を使用する。なお、『テクニカルデータブック』で、カウンタ#2 の動作モードがモード 2 となっているのは誤りで、実際はモード 3 である。

カウンタ#2 を動作モード 3 に設定するには、I/O ポート 77H に B6H を出力すればよい。分周値の設定は I/O ポート 75H に下位バイト、上位バイトの順で出力するだけでよい。通信速度から分周値を求める方法は後述する。動作モードと分周値を設定するプログラム例を List 2 に示す。

List 2　分周値の設定

```
; 分周値の設定（CX レジスタ:設定値）
        pushf
        cli
        mov     al,0b6h                 ; カウンタ＃２をモード３に設定する
        out     77h,al                  ; 動作モードを設定
        jmp     $+2                     ; リカバリタイム用
        jmp     $+2
        mov     al,cl
        out     75h,al                  ; 分周値の下位バイトを出力
        jmp     $+2                     ; リカバリタイム用
        jmp     $+2
        mov     al,ch
        out     75h,al                  ; 分周値の上位バイトを出力
        popf
```

通信速度は8253に設定する分周値によって決まる。分周値と通信速度の関係はシステムクロック（CPUクロックのことではない）の系列によって異なる（Table 2参照）。

Table 2　8253に設定する分周値（調歩同期1/16モード）

通信速度（bps）	8MHz系	5MHz系
75	1664	2048
150	832	1024
300	416	512
600	208	256
1200	104	128
2400	52	64
4800	26	32
9600	13	16
19200	設定不可	8
38400	設定不可	4

PC-9800シリーズでは、システムクロックには5MHz系と8MHz系があり、システムクロックが5MHz系の場合は8253には2.4576MHzのクロックが、8MHz系の場合は1.9968MHzのクロックが供給されている（システムクロックの詳細については別節の「システムクロックの取得」参照）。

8253は供給されたクロックを指定された分周値で分周して8251用のクロックを作成し、8251に供給している。

現在のシステムクロックが5MHz系か8MHz系かは**システム共通域のBIOSフラグ（BIOS FLAG）0000：0501H**を参照して判定する。プログラム例をList 3に示す。

List 3　システムクロック判別

```
; システムクロック判別
        sub     ax,ax               ; システム共通域のセグメントアドレス:0000H
        mov     es,ax
        mov     al,es:[0501h]       ; システム共通域のBIOS-FLAG
        test    al,80h              ; クロック種別
        jz      clock5              ; 5MHz系の処理へ
        jmp     clock8              ; 8MHz系の処理へ
```

PC-9800シリーズのシリアルポートの仕様上の通信速度は調歩同期モードで75bps〜9600bps (75bps×2^n)であるが、システムクロックが5MHz系の場合は、8253を直接制御することで19200bpsや38400bpsが使用できることが知られている。

38400bpsはPC-9800シリーズのハードウェア構成では8251の動作クロックと送受信クロックとの関係で動作範囲外となることがあるので、マージン低下や誤動作の可能性がないとはいえない。したがって、38400bpsを使用する場合は安定して動作をすることを確かめる必要がある。また、PC-9801RA21などでカスタムチップ内に組み込まれている8251は従来の8251より高速であるらしく、8253に設定する分周値を2にすると8251が76800bpsで動作する。

システムクロックが8MHz系の場合は8253への供給クロックと8253の分周値の問題で19200bpsや38400bpsを使用することがハードウェア的にできない。たとえば、PC-H98シリーズは、CPUは高速だがシステムクロックが8MHz系であるため、本体のシリアルポートを調歩同期式の19200bpsや38400bpsで使用することはできない。

3 キャラクタ長、パリティ、ストップビット長などを設定する

キャラクタ長、パリティ、ストップビット長などはシリアルインターフェイスの8251に設定する。8251はシリアルポートの中核をなすLSIで、シリアルデータとパラレルデータの変換やパリティビットの作成、パリティビットやストップビットの検査、各種の制御信号のチェックをしている。

シリアルポートの送受信処理は主に8251に対する操作で行われる。I/Oポート32Hで8251の動作モードなどを設定し、I/Oポート30Hで送受信するデータの入出力をする。8251だけではシリアルポートに必要な機能をすべてカバーできないので、これを補う形で8253や8255が使用されている。

8251では多様な通信方式が設定できるが、実用上はTable 3に示す4種類をサポートするだけで十分である。8251の詳細については、8251のデータシートなどを参照してもらいたい。

Table 3　通信方式の種類

キャラクタ長	パリティ
8ビット	なし
7ビット	偶数
7ビット	奇数
7ビット	なし

通信で日本語やバイナリデータを扱う場合はキャラクタ長が8ビットでパリティなしという設定が普通であるが、海外のシステムや古いシステムなどでは7ビット長も使用されている。

8251は初期化時のリカバリタイムがとても大きいので、初期化の際には十分な時間間隔をおい

て8251にコマンドを出力する必要がある。8251のリカバリタイム用のウェイト処理のプログラム例をList 4に示す。

List 4　8251初期化時のリカバリタイム用ウェイト

```
; 8251初期化時のリカバリタイム用
RsWait  PROC
        mov     cx,16           ; 次のloop命令を16回実行
        loop    $
        ret
RsWait  ENDP
```

8251には3バイトのコマンドやデータを受け付けるので、8251がどのような状態になっていても確実に設定を行うためには、実際に設定したいコマンドの前にダミーのコマンドを3回発行し、そのあとに8251をソフトウェア的にリセットしてから設定を行うようにする。ダミーコマンドには、8251のデータシートなどのプログラム例では00Hが使用されている。

List 5　8251のモード設定

```
; 8251のモード設定
        mov     al,0            ; ダミーコマンド:00H
        out     32h,al          ; ダミーコマンド発行
        call    RsWait
        out     32h,al          ; ダミーコマンド発行
        call    RsWait
        out     32h,al          ; ダミーコマンド発行
        call    RsWait
;
        mov     al,01000000b    ; 8251リセット
        out     32h,al
        call    RsWait
;
        mov     al,00000010b    ; ×16モード
        mov     bx,datalen
        or      al,rscmd_d[bx]  ; データ長の設定
        mov     bx,parity
        or      al,rscmd_p[bx]  ; パリティの設定
        mov     bx,stopbit
        or      al,rscmd_s[bx]  ; ストップビット長の設定
        out     32h,al          ; モード命令を出力
        call    RsWait
```

List 5 のプログラム例では、プログラムのデータ部に List 6 のようなデータがあることを前提にしている。

List 6　初期化パラメータ

```
rscmd_d db      00001000b           ; 0: キャラクタ長 7bit
        db      00001100b           ; 1: キャラクタ長 8bit

rscmd_p db      00110000b           ; 0: 偶数パリティ
        db      00010000b           ; 1: 奇数パリティ
        db      00000000b           ; 2: パリティ無し

rscmd_s db      01000000b           ; 0: ストップビット 1bit
        db      10000000b           ; 1: ストップビット 1.5bit
        db      11000000b           ; 2: ストップビット 2bit

siocmd  db      00110111b           ; 8251 に出力したコマンドの保存用
```

8251 の初期化処理中は ER 信号が一時的に OFF になる。通常のモデムは ER 信号が一定時間以上 OFF のままだと回線を切ってしまう。そこで、通信中に 8251 の再初期化を行いたい場合は、ER 信号が OFF になる時間を極力短くしなければならない。

そこで、通信中の再初期化処理では、ER 信号が OFF になる時間を増やしている原因となるダミーコマンドの発行を省くようにする。

4 制御信号線の状態を設定する

シリアルポートの初期化後は、信号線の ER 信号と RS 信号を ON にするのが一般的である。8251 の初期化が終了すると、8251 はコマンドを受け取れる状態になっているので、8251 のコマンドレジスタ（I/O ポート 32H）に 37H を出力して、ER 信号と RS 信号を ON にする。プログラム例を List 7 に示す。制御信号線については、別節の「RS-232C 信号線の制御」であらためて解説する。

List 7　信号線の状態の初期設定

```
; 信号線の状態の初期設定
        mov     al,37h          ; 送受信許可、ER 信号と RS 信号は ON
        mov     siocmd,al       ; 後々のために出力値を変数 siocmd に保存
        out     32h,al          ; コマンドを出力
```

5 割り込みコントローラの割り込みマスクを設定する

シリアルポートをハードウェア割り込みを使って使用する場合は、割り込みコントローラ 8259 の制御が必須であり、そのための設定を行わなくてはならない。実際には、8259 の **IMR (割り込みマスクレジスタ) である I/O ポート 00H** の bit4 でシリアルポートの割り込み信号を禁止するかどうかを設定する。

8259 はシステム起動時に初期化されているので、シリアルポートを使用する場合は割り込みマスクの設定を変えるだけで十分で、8259 全体の再初期化を行う必要はない。8259 の再初期化は、すべてのハードウェア割り込みの設定を詳細に知っている場合に限られる。

プログラムを List 8 に示す。

List 8　シリアルポートからの割り込みマスクを設定

```
; シリアルポートからの割り込みを許可
        cli
        in      al,02h          ; 割り込みコントローラから IMR を読む
        and     al,11101111b    ; 割り込みマスクを解除
        out     02h,al
```

6 シリアルポート用の割り込みベクタを保存、割り込み処理を登録する

シリアルポートの初期化では 8251、8255、8253、8259 の設定をし、ハードウェア割り込みを使用するためには割り込みベクタテーブルへ割り込み処理を登録する必要がある。シリアルポートの割り込みは**ベクタ番号 0CH** を使用する。

実際のアプリケーションプログラムでは、自前の割り込み処理を登録する前に割り込みベクタテーブルに登録されていたベクタの保存も行い、プログラム終了時に元に戻す必要がある。これを怠ると、アプリケーションプログラムが終了すると割り込みベクタが指しているアドレスはプログラムが存在しないアドレスを指していることになり、ハードウェア割り込みが発生した途端にシステムが暴走してしまうことになる。この保存先のアドレスを rsivseg と rsivofs とする。

割り込みベクタの保存と設定を行うプログラムを List 9 に示す。

割り込み処理は List 10 のように用意されているものとする。割り込み処理については、「割り込み編」であらためて解説する。

List 9　割り込みベクタの保存と設定

```
; 割り込みベクタの保存と設定
        mov     ax,350ch            ; INT-0CH ベクタの設定を取得
        int     21h
        mov     rsivseg,es          ; セグメントアドレスを保存
        mov     rsivofs,bx          ; オフセットアドレスを保存
;
        push    ds
        mov     ax,cs
        mov     ds,ax
        mov     dx,OFFSET RsIntrEntry   ; 割り込み処理のアドレス
        mov     ax,250ch            ; INT-0CH ベクタに割り込み処理を登録
        int     21h
        pop     ds
```

List 10　割り込み処理

```
; 割り込み処理
RsIntrEntry PROC FAR
        ; ここに割り込み処理が入る
        iret
RsIntrEntry ENDP
```

また、ワークエリアもList 11のように用意する。

List 11　ワークエリア

```
; ワークエリア
rsivadr label   dword
rsivofs dw      ?                   ; ベクタ保存用
rsivseg dw      ?                   ; ベクタ保存用
```

こうしておけば、プログラム終了時など、シリアルポートをクローズしたときに割り込みベクタを元の処理に戻すことができる。割り込みベクタの復帰プログラムをList 12に示す。

List 12　割り込みベクタを復帰

```
; 割り込みベクタを復帰
        push    ds
        lds     dx,rsivadr
        mov     ax,250ch            ; INT-0CH ベクタを元に戻す
        int     21h
        pop     ds
```

7 シリアルポートからの割り込みを許可する

シリアルポートの割り込み許可は、割り込み禁止と同様の手法を用いて、シリアルポート関係のビットのみを許可に変化させる。プログラムをList 13に示す。

List 13　RS-232C の送受信割り込みを許可する

```
; RS-232C の送受信割り込みを許可する
        pushf
        cli
        in      al,35h              ; システムポートを読む
        jmp     $+2                 ; リカバリタイム用
        and     al,11111000b
        or      al,00000101b        ; 送受信割り込みを許可
        out     35h,al
        popf
```

▶ワンポイント

■8251の外部同期式の動作

8251では調歩同期式だけではなく同期式を使用することもできる。しかし、そのためにはPC-9800シリーズの場合はディップスイッチを外部同期式に設定して、シリアルポートの送受信動作が外部クロックによって行われるようにしなければならない。

シリアルポートに接続する機械の大部分は調歩同期式が使われており、同期式が接続相手を選ばない汎用のシリアルインターフェイスに使用されることはないので、本章では同期式の通信は解説していない。調歩同期式と同期式では8251の動作がまったく異なり、同期式ではスタートビットやストップビットは存在しない。

■8251の調歩同期式の×16モードと×1モード

8251を調歩同期モードで使用する場合は×16という動作モードで使用するが、これを×1モードで使用すればシステムクロックが8MHz系でも19200bpsや38400bpsの設定が可能になる。しかし、調歩同期式として動作するにはサンプリング動作のためのマージンが確保できないので、送信はできるが正しく受信できないという症状が発生する。×1モードは、外部からクロックを入力する場合に使う特殊なものである。

■8251の受信許可ビット操作上の注意

8251のコマンドレジスタには送信許可や受信許可のビットがあるが、受信許可のビットを変更する場合は注意が必要である。8251の受信許可ビットは受信動作自体の許可や禁止を行うものなので、通常は常に受信許可にしておく。受信データを一時的に無視したい場合はシステムポートの8255を操作して受信割り込みを禁止するようにする。

8251のコマンドレジスタで受信禁止を指定すると、つぎに受信許可した時点からスタートビットの検出が始まる。このため、連続して受信データが送られてきている場合は同期外れ（データの一部をスタートビットと誤認する現象）が発生し、データ化けが発生することがある。

8255へのコマンドで受信割り込みの禁止を行った場合は、禁止を解除するまで受信データが消失するだけで、同期外れが発生することはない。

■拡張RS-232Cボードを扱う上での注意点

NECのPC-9861K（拡張RS-232Cボード）はPC-9800本体のRS-232Cとは構造が異なり、通信速度をボード上のディップスイッチで設定するようになっている。ソフトウェアからは通信速度を設定できない。このため、拡張スロットにボードを差し込んだまま通信速度を自由に変えることはできない。

この通信速度のディップスイッチの設定は、ボードに付属のマニュアルでは75bps～9600bpsの8種類しか公開されていないが、設定を変えることで19200bpsが使用できることが知られている。PC-9800シリーズの拡張スロットには19200bps用のクロックが用意されていて、PC-9861Kは、このクロックを利用しているからである。このため、38400bpsは設定できない。

■システムクロックと転送速度

システムクロックが8MHz系の場合にシリアルポートで19200bpsや38400bpsが使用できないのは、シリアルインターフェイス用LSIの8251が原因ではなく、PC-9800シリーズのクロック関係の回路の設計が悪いためである。

キーボードインターフェイスで使用している8251は19200bpsでキーボードと通信している。これはシステムクロックが8MHz系であるためにシリアルポートで19200bpsが使用できない機種でも同じである。

また、システムクロックが8MHz系の機種でも拡張スロットにはキーボードインターフェイスで使用している19200bps用のクロックが出力されている。

■高速転送の可能性

シリアルポートの転送速度は現在のCPUの能力に比べるとかなり低速である。シリアルインターフェイスの8251の機能を使用する通常の利用方法（調歩同期式）では9600bpsや38400bps程度が限界である。

しかし、8251の機能を使用せずに、RS-232Cの制御信号線でデータを送受信できるように接続ケーブルを作成し、独自のプログラム同士で通信すれば、はるかに高速な通信を行うことも可能である。

ライブラリ

RsOpen

解説 RS-232Cの割り込み処理の登録とRS-232Cの初期化を行う。書式はつぎのとおり。

void **RsOpen**(int *speed*, int *datalen*, int *parity*, int *stopbit*, int *flow*)

speed 通信速度

数　値	ボーレイト
0	75
1	150
2	300
3	600
4	1200
5	2400
6	4800
7	9600
8	19200
9	38400

▶▶ CONTINUED

datalen データ長

数値	意味
0	7bit
1	8bit

parity パリティ

数値	意味
0	Even
1	Odd
2	None

stopbit ストップビット

数値	意味
0	1bit
1	1.5bit
2	2bit

flow フロー制御

数値	意味
0	なし
1	XON/XOFF
2	RS/CS

初期化により RS-232C の ER 信号と RS 信号は ON になる

戻り値 なし

サンプル TERM.C

SetSpeed

解説 RS-232C の通信速度を設定する。書式はつぎのとおり。

int **SetSpeed**(int *speed*)

speed 通信速度。値とボーレイトの関係は RsOpen 参照

戻り値 なし

サンプル TERM.C

▶▶ CONTINUED

RsReOpen

解説　RS-232C のデータ長などの設定を変更する。書式はつぎのとおり。

void **RsReOpen**(int *datalen*, int *parity*, int *stopbit*, int *flow*)

datalen　データ長

数値	意味
0	7bit
1	8bit

parity　パリティ

数値	意味
0	Even
1	Odd
2	None

stopbit　ストップビット

数値	意味
0	1bit
1	1.5bit
2	2bit

flow　フロー制御

数値	意味
0	なし
1	XON/XOFF
2	RS/CS

戻り値　なし

サンプル　TERM.C

RsClose

解説　RS-232Cの割り込みの停止とベクタの登録解除を行う。書式はつぎのとおり。

void **RsClose**(void)

RS-232C の ER 信号と RS 信号は OFF になる

戻り値　なし

サンプル　TERM.C

シリアルポートの送受信割り込み処理

ここでは送受信割り込みによってシリアルポートの入出力をするルーチンを解説する。このルーチンは、シリアルポートの初期化の際にシリアルポート用の割り込みベクタに登録する。シリアルポートで割り込みが発生すると、このルーチンが受信データをバッファに蓄えたり、送信バッファからデータを送信したりする。

実際にプログラムからデータを送ったり受信したりするプログラムは、「シリアルポートからのデータの受信」「シリアルポートへのデータの送信」の節でとりあげる。

▶ポイント

- PC-9800シリーズでは送信と受信で1つの割り込みを共有するようになっている。
- 割り込みルーチンの最初でI/Oポートを操作して送受信割り込みを禁止する。
- 1つの割り込みルーチンの中で受信と送信の両方を処理する。
- 受信処理では受信したデータを受信バッファに格納する。
- 送信処理では送信バッファから取り出したデータを送信する。
- 送信処理のあとに送信バッファが空になっていれば送信割り込みを禁止する。
- 割り込みルーチンは、実行時間が可能な限り短くなるようにする。

▶プログラミングテクニック

1 送受信割り込みを禁止する

割り込みコントローラは割り込み信号の立ち上がりエッジで動作するため、送受信割り込みを使用する場合は送信と受信がほぼ同時に発生した場合でも正しく送受信ができるように作成する必要がある。これは、割り込み処理内で一時的に送受信割り込みを禁止することで対応できる。

シリアルポートの割り込みを禁止するには、つぎのように3つの方法がある。

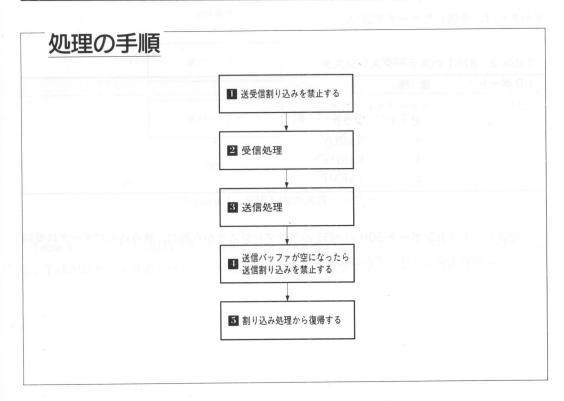
- I/Oポート35H（8255のポートC）のTXRE、RXREビットを変更
- I/Oポート32H（8251のコマンドレジスタ）のTXE、RXEビットを変更

● I/O ポート 02H（8259 の割り込みマスクレジスタ）の M4 ビットを変更

ここで行いたいのは、送信割り込み処理中に受信割り込みが発生した場合や、受信割り込み処理中に送信割り込みが発生した場合に対応するための割り込み禁止なので、I/O ポート 35H で 8255 を操作する方法が適している。

I/O ポート 32H の 8251 のコマンドレジスタの RXE ビットを変更する方法では受信動作自体が禁止されるため、恒久的な受信の禁止には使用できるが、割り込み処理内での禁止には適さない。I/O ポート 02H で 8259 の割り込みマスクレジスタを変更する方法は、完全にシリアルポートの使用を止めるときなどに使用する。I/O ポート 35H は Table 1 のようになっている。これを操作して送受信割り込みを禁止する方法は、List 1 のとおりである。

Table 1　I/O ポート 35H（8255 のポート C）

I/O ポート	意味			
35H	8255 のポート C			
	ビット	信号	値	意味
	bit0	RXRE	0	受信割り込みの禁止
			1	受信割り込みの許可
	bit2	TXRE	0	送信割り込みの禁止
			1	送信割り込みの許可

List 1　送受信割り込みを禁止

```
RsIntrEntry PROC FAR
        :
        :                               ; レジスタ待避
        in      al,35h                  ; 送受信割り込みのマスクを読む
        jmp     $+2                     ; リカバリタイム用
        push    ax                      ; 割り込みマスク状態を保存
        and     al,11111000b            ; 送受信割り込みを禁止
        out     35h,al
```

2 受信処理

PC-9800 シリーズのシリアルポートでは、送受信割り込みを使用する場合、送信と受信で1つの割り込みを共有するようなハードウェア構成になっている。そのため、シリアルポートの割り込み処理では割り込みの要因が送信なのか受信なのかを判別する必要がある。ここでは、最初に受信の処理をしてから送信の処理をすることにする。

送信割り込みの処理中にデータを受信したり、受信割り込みの処理中に送信が可能になったりする場合については、割り込み処理の最初に割り込み禁止を行うことで解決している。

受信では、I/O ポート 32H の 8251 のステータスをチェックし（Table 2 参照)、受信したデータがあれば、受信したデータを読む。

Table 2　8251 のステータスレジスタ

I/O ポート	意　味	
32H	ステータスレジスタ	
	ビット	フラグ
	0	TxRDY
	1	RxRDY
	2	TxEMP

受信データは I/O ポート 30H の 8251 のデータレジスタから読む。読み込んだデータは受信バッファに格納する。プログラムは List 2 のようになる。

List 2　受信処理

```
          in       al,32h          ; 8251 のステータスをチェック
          test     al,00000010b    ; 受信データがあるか
          jz       rsout
          in       al,30h          ; 受信データを読む
          :                        ; 受信データを受信バッファに格納する
rsout:
```

3 送信処理

受信の処理が終わったら送信の処理をする。まずI/Oポート32Hの8251のステータスをチェックし、送信が可能かどうかを調べる(Table 2参照)。送信が可能で、送信すべきデータがあれば、送信バッファから読み出したデータを送信する。

送信はI/Oポート30Hの8251のデータレジスタに書き込むことで行う。

8251はFigure 1のように、内部に第1バッファと第2バッファという2個のレジスタを持っている。第1バッファは、送信データの送信（パラレル→シリアル変換）を行うレジスタである。第2バッファは、CPUからの書き込みで使用されるデータレジスタである。このレジスタの状態は、8251のステータスレジスタのTxRDYビットとTxEMPビットで確認することができる。

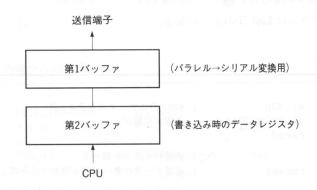



Figure 1　8251内の送信バッファ

Table 3のように、TxRDYビットは第2バッファが空のときに1になり空でないときは0になる。TxEMPビットは第1バッファが空のときに1になり空でないときに0になる。

Table 3　8251 内の送信バッファとステータスの関係

第 1 バッファ 第 2 バッファ	空 空	データ 1 空	データ 1 データ 2	データ 2 空	空 空
TxRDY ビット	1	1	0	1	1
TxEMP ビット	1	0	0	0	1

第 1 バッファと第 2 バッファがともに空のときに、CPU が 8251 に送信データを書き込むと、送信データは即座に第 1 バッファに転送され、第 2 バッファは空になり、送信が開始される。このとき、TxEMP ビットは 0 になる。

最初のデータの送信が終わる前に CPU がデータレジスタに送信データを書き込むと、送信データは第 2 バッファに留まり、TxRDY ビットも 0 になる。

最初のデータが送信し終わると、第 2 バッファのデータが第 1 バッファに自動的に転送され、第 2 バッファは空になり、TxRDY ビットが 1 に戻る。このあと、CPU は送信データをデータレジスタに書き込むことが可能になる。ここで書き込みを行わず、2 番目の送信が終わると、TxEMP ビットも 1 に戻る。

8251 の能力を活かすため、送信可能かどうかのチェックは TxEMP ビットではなく TxRDY ビットで行う。システムポートの 8255 でシリアルポートの送信割り込みを TXEE ビットではなく TXRE ビットで行っているのも、8251 の能力を活かすためである。

送信処理のプログラムは List 3 のようになる。

List 3　送信処理

```
        in      al,32h          ; 8251のステータスをチェック
        test    al,00000001b    ; 送信が可能か
        jz      rsend
        :                       ; 送信バッファを調べる
        jz      rsmask          ; 送信データが無ければ送信割り込み禁止へ
        :                       ; 送信データを送信バッファから読み出す
        out     30h,al          ; データを送信
```

4 送信バッファが空になったら送信割り込みを禁止する

送信処理を行ったあとに送信バッファが空になっていれば、送信割り込みを禁止する必要がある。送受信割り込みの禁止は割り込み処理の最後で行うようにしているため、ここでは割り込み禁止の指定をする。プログラムはList 4のようになる。

List 4　送信バッファが空になったら送信割り込みを禁止

```
:                                   ; 送信バッファを調べる
        jnz     rsend               ; 送信データがあれば送信割り込み禁止はしない
rsmask:
        pop     ax
        and     al,11111001b        ; 送信割り込み禁止の指定
        push    ax
rsend:
```

5 割り込み処理から復帰する

最初に送受信割り込みを禁止したので、それを解除する。4で送信割り込み禁止のマスクを指定した場合は、それも設定する。プログラムはList 5のようになる。

List 5　割り込み処理からの復帰

```
        mov     al,00100000b        ; EOI コマンド
        out     00h,al              ; 割り込みコントローラに EOI を出力
        pop     ax
        out     35h,al              ; 送受信割り込みのマスクを設定
        :                           ; レジスタ復帰
        iret
RsIntrEntry ENDP
```

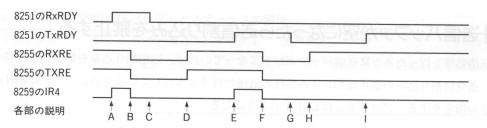
以上が送受信割り込み処理ルーチンである。実際に送信や受信が発生した場合は、Figure 2のようなタイムチャートになる。上記のような構造のプログラムであれば、送信や受信がどのようなタイミングで発生しても確実にハードウェア割り込みが発生して、正しく送信や受信を行うことができる。

A：データ受信により8251から受信可フラグが立ち、割り込み処理が起動する
B：送受信割り込みを8255の設定で一時的にマスクする
C：8251から受信データを読み出すと自動的に受信可フラグが降りる
D：割り込み処理の終了直前に8255の設定で割り込みのマスクを解除する
E：データ送信可能になったので、8251の送信可フラグが立ち、割り込み処理が起動する
F：送受信割り込みを8255の設定で一時的にマスクする
G：送信データを8251に出力すると自動的に送信可フラグが降りる
H：つぎに送信すべき文字がないので、受信割り込みのみマスクを解除する
I：送信可になるが送信割り込みはマスクされているので、割り込み処理は起動しない

Figure 2　送受信割り込みのタイムチャート

▶ワンポイント

■なぜ割り込みを使うのか？

シリアルポートの送受信は、単純に考えるとシリアルポートからデータを読み込む処理とシリアルポートにデータを送り出す処理を交互に繰り返せば良いように思うかもしれない。しかし、シリアルポートにデータが到着するタイミングは非常に速く、ディスクアクセスや画面表示といった時間のかかる処理の間にそうした処理を行っていたのではデータを取りこぼしてしまう。

そこで、通常はシリアルポートの受信割り込み機能を利用する。受信割り込みが発生した場合、とりあえず受信したデータをバッファに蓄え、必要に応じて別の処理でバッファから取り出すという処理を行う。

RS-232C BIOS や MS-DOS の RSDRV.SYS も受信に関してはこのような処理を行っているが、送信に関してはソフトウェアのループでポーリングしているだけである。しかし、実際には XON/XOFF によるフロー制御などで迅速な送信が必要な場合もあり、送信割り込みも利用した方がよい。送信割り込みを使用すると転送効率が向上する。

■割り込み処理と実行時間

割り込みルーチンは割り込み禁止状態で実行が開始されるため、割り込みルーチン内ではSTI命令を実行しない限りNMI以外のハードウェア割り込みは受け付けられない。シリアルポートの割り込みは他の割り込みと比較して緊急性が高いと考えられるため、割り込み処理の中では割り込みの許可はしない。このため、他の割り込み処理などの迷惑にならないように、可能な限り実行時間が短くなるように注意する必要がある。

たとえば通信速度が38400bpsともなれば、シリアルポートの割り込みは受信だけでも約250μ秒に1回の割合で発生する。送受信割り込みを使用している場合は、送信と受信がともに行われていると、約2倍の頻度で割り込みが発生することになる。

1回の割り込み処理時間は割り込み発生の間隔より十分に短くなくてはならない。ディスク入出力などの最中はDMA転送によってCPUの動作が妨げられ、CPUの見かけ上の実行速度が低下してしまうので、ハードディスクのDMA転送中などでは割り込みルーチンの実行時間もかなり長くなる。

また、割り込みルーチンの実行時間がCPUの実行時間を占有しても困るため、割り込みルーチンには非常にわずかな実行時間しか与えられないことになる。シリアルポートの割り込みルーチンは、このような理由から、実行時間が可能な限り短くなるように工夫をする必要がある。

■送受信割り込みに適さない場合

送受信割り込みを使用するプログラムは、受信割り込みとポーリングで送信を行っているプログラムより割り込み処理などが複雑になる。そのため、低速のCPUを使用して高速に受信のみを行うことが中心となるような用途や、半二重通信など送受信が同時に発生しない用途には適さないこともある。

シリアルポートからのデータの受信

シリアルポートとのデータのやり取りは、先の送受信割り込みの処理ルーチンの中ですべて行われている。したがって、送受信割り込みを利用してシリアルポートからデータを受信する場合は、受信関数自身は割り込み処理ルーチンの受信バッファからデータを取り出すだけである。また、受信関数は割り込みに関する処理を行う必要もない。

ここでは、その受信関数を解説する。なお、ライブラリには受信バッファのデータを調査する関数もあるが、受信関数内のバッファの空きを調べる部分を独立させるだけである。

▶ポイント

- データは、送受信割り込みの処理によって通信を行うソフトウェアと非同期に受信バッファに格納されているので、受信バッファからデータを読み出す。

▶プログラミングテクニック

1 受信バッファから読み出す

シリアルポートを送受信割り込みで使用している場合は、送受信処理は受信バッファや送信バッファに対する処理が中心となる。したがって、8251のポートを監視するのではなく、受信の場合は受信バッファの中にデータが入っているかどうか、送信の場合は送信バッファに十分な空きがあるかどうかで判断する。

RS-232C BIOSではステータス信号などもデータとしてバッファリングしていて、1回の受信で2バイトを消費するが、実際にステータス信号などをバッファリングして記録しておく必要性はほとんどない。このため、受信バッファは1回の受信で1バイトを使用する単純なリングバッファとし、メモリを効率良く使用するようにする。

受信バッファから読み出すプログラムはList 1のようになる。ただし、実際のプログラムではフロー制御関係の処理が入るため、これより複雑になっている。

List 1　受信バッファから1データを読み出す

```
        pushf
        cli
        mov     ax,-1               ; 受信データがないときの戻り値は-1
        cmp     rblen,0
        jz      rdend
        mov     bx,rbrdpos          ; 読み出し位置
        mov     al,rbbuf[bx]        ; 受信データ読み出し
        sub     ah,ah
        inc     rbrdpos
        dec     rblen
        cmp     bx,RBSIZE           ; バッファの終端か
        jb      rdend
        mov     rbrdpos,0           ; 読み込み位置を受信バッファの先頭へ
rdend:
        popf
```

▶ワンポイント

■バッファのサイズ

受信バッファの大きさは、通信速度とフロー制御の反応速度によって決める。通信速度が低くてフロー制御の反応が速ければ小さな値にできるが、通信速度が高くてフロー制御の反応が遅ければ大きな値にしなければならない。

当ライブラリではバッファの大きさは受信バッファを約6Kバイトに、送信バッファを約2Kバイトに固定してある。通常の利用方法では、この程度の大きさで十分である。

処理の手順

1 受信バッファから読み出す

ライブラリ

ReceiveData

解説　受信バッファから１文字を取得する。書式はつぎのとおり。

int **ReceiveData**(void)

戻り値　取得した文字。受信したデータがないときは−1になる。

サンプル　TERM.C

ReceiveLength

解説　受信バッファ内のデータの文字数を取得する。書式はつぎのとおり。

int **ReceiveLength**(void)

戻り値　受信バッファ内のデータの文字数。

サンプル　TERM.C

ReceiveSpace

解説　受信バッファ内の空き文字数を取得する。書式はつぎのとおり。

int **ReceiveSpace**(void)

戻り値　受信バッファ内の空き文字数。

COLUMN

文字落ちの原因

シリアルポートを使用して高速な通信を行うとき、しばしば文字落ちが問題となる。通信ソフトウェアの処理速度が原因となっている場合は適切なフロー制御を行うことで回避できる。しかし、文字落ちは原因が通信ソフトウェア以外にある場合が多く、回避がむずかしいこともある。

通常、シリアルポートによる通信では割り込みを使用してデータを受信する。そのため、割り込みが長時間禁止されてしまうとデータを取りこぼし、いわゆる文字落ちが発生する。通信ソフトウェアは当然ながら割り込み禁止時間が長くならないように考慮して作られている。しかし、通信に関係のないプログラムは、パソコン通信が一般化する以前は、割り込み禁止時間について考慮していないものが多かった。

実際に文字落ちの原因となったプログラムには、日本語入力FEP、RAMディスクドライバ、ディスクキャッシュ、EMSドライバ、各種TSRなどがあった。これらのソフトの多くは、バージョンアップによって問題を解決した。また、ファームウェアでは、PC-9801互換機のROM BIOS、サードパーティ製のハードディスクインターフェイス上のディスクBIOSなどが、やはり割り込み禁止時間を考慮していなかったことやバグにより、文字落ちの原因となったことがあった。このようなファームウェアでは、ROMを交換して解決することになった。

割り込み処理ルーチンを作る者として、処理速度を考慮することを常に肝に命じて置くべきである。

シリアルポートへのデータの送信

シリアルポートとのデータのやり取りは、先の送受信割り込みの処理ルーチンの中ですべて行われている。したがって、送受信割り込みを利用してシリアルポートにデータを送信する場合は、送信関数自身は、割り込みの処理ルーチンの送信バッファにデータを加え、データが送信されるように送信割り込みが発生するようにするだけである。

ここでは、実際に送信データをバッファに格納し、割り込みを有効にする送信関数を解説する。

▶ポイント

- 送信するデータは送信バッファに格納する。送信バッファに空きがない場合は、そのまま終了する。したがって、この送信関数を呼ぶ前に、バッファに空きがあるかどうか確認をしなければならない。
- データを送信バッファに格納したら、送信割り込みを許可する。実際の送信は送受信割り込み処理ルーチンに任せる。
- 送信バッファの空き状態を調べる関数を、別に用意する。ここでは特に内部を解説しないが、送信関数のバッファの空きを確認する部分を独立させるだけである。

▶プログラミングテクニック

1 送信バッファにデータを格納する

データを送信する場合は送信バッファにデータを格納し、送信割り込みの禁止を解除するだけである。あとは送信割り込みが発生したときに、割り込み処理ルーチンが送信処理を自動的に行ってくれる。

送信バッファにデータを格納するプログラムはList 1のようになる。

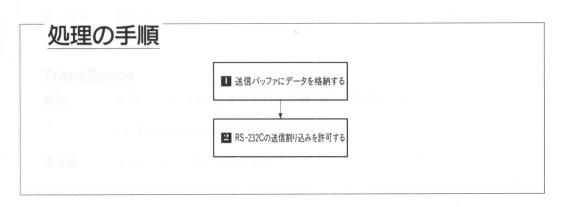
List 1　送信バッファへのデータの格納

```
        cmp     tblen,TBSIZE        ; 送信バッファに空きがあるか
        jae     txend
        mov     bx,tbwrpos
        mov     ax,txdata
        mov     tbbuf[bx],al        ; データを送信バッファに書き込む
        inc     tbwrpos
        inc     tblen
        cmp     bx,TBSIZE           ; バッファの終端か
        jb      send
        mov     tbwrpos,0           ; 書き込み位置を受信バッファの先頭へ
send:
```

2 RS-232C の送信割り込みを許可する

送信割り込みの起動は送信割り込みの禁止を解除するだけである。送信割り込みの許可はI/Oポート35Hを1ビット変更するだけだが、このような場合は**I/Oポート37H**の8255のビットセット命令やビットリセット命令を使用する方が簡単である（Table 1 参照）。

Table 1　8253 のビットセット／ビットリセット

I/Oポート	意　味	
37H	8253 のモードセット	
	値	意　味
	0	RXRE の OFF
	1	RXRE の ON
	2	TXEE の OFF
	3	TXEE の ON
	4	TXRE の OFF
	5	TXRE の ON

プログラムはList 2のようになる。

List 2　送信割り込みの禁止の解除

```
        mov     al,00000101b    ; RS-232Cの送信割り込みを許可
        out     37h,al
txend:
```

これは、List 3のような処理と同じ動作をする。

List 3　送信割り込みの禁止の解除

```
        pushf
        cli
        in      al,35h
        jmp     $+2
        or      al,00000001b    ; RS-232Cの送信割り込みを許可
        out     35h,al
        popf
```

送受信割り込みを使っている場合は、シリアルポートからの割り込み処理でポート35Hの値が変わることがあるので、この部分は割り込み禁止状態にする必要がある。ポート37Hでビットセット命令やビットリセット命令を使用している場合は単一のOUT命令で済むため、このような割り込み禁止は不要になっている。

ライブラリ

TransData

解説　1文字を送信バッファに入れて送信する。書式はつぎのとおり。

void **TransData**(int *txdata*)

txdata　送信する1バイトのデータ

戻り値　なし

サンプル　TERM.C

TransLength

解説　送信バッファ内のデータの文字数を取得する。書式はつぎのとおり。

int **TransLength**(void)

戻り値　送信バッファ内のデータの文字数

サンプル　TERM.C

TransSpace

解説　送信バッファ内の空き文字数を取得する。書式はつぎのとおり。

int **TransSpace**(void)

戻り値　送信バッファ内の空き文字数

RS-232C信号線の制御

RS-232Cのコネクタには、データを送ったり受けたりする信号線以外にもいくつかの制御用の信号線が用意されている。これらを制御すれば、接続先の装置をうまく扱うことができる。

本節ではこれら制御用信号線の処理を各信号線ごとに解説する。各信号線の処理は独立しているので、関数も各信号線ごとに分けている。

▶ポイント

- 出力信号にはER信号とRS信号がある。
- 入力信号にはCS信号とCD信号とDR信号とCI信号がある。
- ER信号は主に回線切断に使用する。
- RS信号は主にハードウェアフロー制御用に使用する。
- CS信号がOFFの状態ではハードウェア的に送信が止まるようになっている。
- CD信号はモデムがキャリアを検出したときにONとなる。
- DR信号は回線が接続されたときにONとなるようにして使うことが多い。
- CI信号はモデムが着信したことを表すのに使用されることが多い。

▶プログラミングテクニック

1 信号線の種類

RS-232Cには、Table 1に示すような制御信号線が用意されている。RS-232Cは、モデムとの接続を想定した規格なので、各信号線の意味はモデムの動作と深く関っている。

それぞれの制御信号線は、シリアルポートの8251（I/Oポート32H）にコマンドを送ることで設定でき、8251のステータスレジスタ（I/Oポート32H）やシステムポートの8255（I/Oポート33H）で監視できるようになっている。

Table 1　RS-232C の制御信号線

信号名	略　号	入出力	信号線の一般的な用途
データ端末レディ	ER	出力	OFF で回線切断 回線制御用で、98 側が通信の用意ができている時に ON にし、回線切断時や通信の用意ができていない時に OFF にする。
送信要求	RS	出力	ハードウェアフロー制御（RS/CS 制御） データ受信の制御用で、データを受信できる状態の時に ON にし、受信バッファがいっぱいで受信できない時や通信の用意ができていないときに OFF にする。したがって、フロー制御に利用できる。
送信可	CS	入力	ハードウェアフロー制御（RS/CS 制御） データ送信の制御用で、ON の時は送信できるが OFF の時は送信できない。CS 信号が OFF になると、ハードウェア的に送信が止まるようになっている。
受信キャリア検出	CD	入力	キャリア検出状態で ON 回線接続の確認用で、モデムがキャリア信号を検出したとき（受信信号が所定の範囲に入ったとき）に ON となり、そうでないときに OFF になる。
データセットレディ	DR	入力	回線接続状態で ON 回線が接続されてキャリア信号を送受信できる状態になっているときに ON となり、そうでないときは OFF となる。
被呼表示	CI	入力	着信時に ON モデムが着呼信号（リング）を受信した時に ON となり、そうでないときに OFF となる。

2 ER 信号の制御

ER 信号は、モデムに対して通信を開始するとき ON にして、回線を切断するとき OFF にする。ER 信号の ON／OFF は 8251 のコマンドレジスタ（I/O ポート 32H）の bit1 への出力によって行う。プログラム例を List 1 に示す。

ER 信号の制御のためには 8251 のコマンドレジスタの bit1 のみを操作する必要がある。しかし、8251 のコマンドレジスタを読み出すことはできない。そのため、ライブラリ中ではコマンドレジスタへ出力する値をメモリ（変数名を siocmd としてある）に保存するようにしている。

ER 信号の状態を変更したい場合には、変数 siocmd より現在のコマンドレジスタの状態を読み出して、ER 信号に関係する bit1 のみ変更し、その値を I/O ポート 32H に出力する。

コマンドレジスタへの出力値をメモリに保存するという方法は、8251 のコマンドレジスタのように状態が読み出せないレジスタを持った LSI を制御する場合の常套手段である。8251 のコマンドレジスタは割り込み処理内でも変更される可能性があるため、このような処理は一時的に CLI 命令で割り込みを禁止して行う必要がある。

また、現在のER信号の状態を取得する場合は、変数siocmdからER信号に該当するビットを抜き出すだけでよい。ER信号の状態取得の例をList 2に示す。

List 1　ER信号の制御

```
; RS-232CのER信号をONにする
        pushf
        cli
        mov     al,siocmd           ; bit1がER信号の制御ビット
        or      al,00000010b
        out     32h,al
        mov     siocmd,al
        popf

; RS-232CのER信号をOFFにする
        pushf
        cli
        mov     al,siocmd           ; bit1がER信号の制御ビット
        and     al,11111101b
        out     32h,al
        mov     siocmd,al
        popf
```

List 2　ER信号の読み出し

```
; RS-232CのER信号の状態を返す
; 戻り値はAXレジスタ： 0:ER=OFF, 1:ER=ON
        mov     al,siocmd           ; bit1がER信号の状態を表すビット
        shr     al,1
        not     al
        and     ax,1
```

3 RS信号の操作

RS信号は主にハードウェアフロー制御（RS/CSフロー制御）に使用する。ハードウェアフロー制御はRS-232CのRS信号とCS信号で受信や送信の一時停止と再開を行うものである。CS信号がOFFであればシリアルポートからの送信はハードウェアによって自動的に停止し、ONになれば自動的に再開するので、送信停止についてはソフトウェアで特別な対処をする必要はない。

RS信号も8251のコマンドレジスタ（I/Oポート32H）のbit5で状態を設定する。プログラム例をList 3に示す。

8251のコマンドレジスタは直接読み出すことができないため、ER信号と同様にコマンドレジス

タへ出力した値は同時にメモリに保存しておく。メモリに保存された値を利用してRS信号に該当するbit5だけを変更している。

現在のRS信号の状態を取得するプログラムはList 4のようになる。

List 3　RS信号の制御

```
; RS-232CのRS信号をONにする
        pushf
        cli
        mov     al,siocmd           ; bit5がRS信号の制御ビット
        or      al,00100000b
        out     32h,al
        mov     siocmd,al
        popf

; RS-232CのRS信号をOFFにする
        pushf
        cli
        mov     al,siocmd           ; bit5がRS信号の制御ビット
        and     al,11011111b
        out     32h,al
        mov     siocmd,al
        popf
```

List 4　RS信号の状態取得

```
; RS-232CのRS信号の状態を返す
; 戻り値はAXレジスタ： 0:RS=OFF, 1:RS=ON
        mov     al,siocmd           ; bit5がRS信号の状態を表すビット
        and     al,00100000b
        cmp     al,1
        sbb     ax,ax
        inc     ax
```

4 CS信号の取得

CS信号はシステムポート（I/Oポート33H）のbit6で状態を調べることができる。8251は、CS信号がOFFの状態ではハードウェア的に送信が停止するようになっている。ハードウェアフロー制御（RS/CSフロー制御）を使用している場合は連続的な送信（アップロードなど）の最中に一時的にCS信号がOFFとなることがある。しかし、RS-232Cに接続した機器がハードウェアフロー制御を利用していないときは、通信中にCS信号がOFFとなることはないはずである。

非通信時にハードウェアフロー制御を利用していない場合は、RS信号とER信号をONにして

もCS信号がOFFであれば、接続した機器の電源が入っていないかコネクタが抜けているか接続ケーブルが断線していることが考えられる。そこで、CS信号がONになっているはずのときにOFFになっている場合は警告メッセージを表示して注意を促すなどのことができる。

なお、非通信時などでRS信号やER信号をOFFにしているときは、接続する機器や接続ケーブルの種類によってはCS信号が自動的にOFFになるものがある。

CS信号の状態を調べるプログラムはList 5のようになる。

List 5 CS信号の状態取得

```
; RS-232CのCS信号の状態を返す
; 戻り値はAXレジスタ： 0:CS=OFF, 1:CS=ON
        in      al,33h              ; bit6がCS信号の状態を表すビット
        rol     al,1
        rol     al,1
        and     ax,1
```

5 CD信号の取得

CD信号は一般的なモデムではキャリアを検出したとき(モデムが接続先のモデムと交信可能になったとき)にONとなる。これは一般的に、回線が接続されている状態と考えてよい。CD信号がOFFであれば、回線は接続されていない状態か、ダイヤル後の接続処理中である。

通信ソフトなどで回線が完全に接続されたかどうかを調べたい場合はCD信号を利用するのが簡単である。ただし、モデムによっては常にCD信号をONにできるので、どんな場合でもこの方法が有効なわけではない。

CD信号はシステムポート（I/Oポート33H）のbit5で調べられる。List 6にプログラムの例を示す。

List 6 CD信号の状態取得

```
; RS-232CのCD信号の状態を返す
; 戻り値はAXレジスタ： 0:CD=OFF, 1:CD=ON
        in      al,33h              ; bit5がCD信号の状態を表すビット
        and     al,00100000b
        cmp     al,1
        sbb     ax,ax
        neg     ax
```

6 DR 信号の取得

DR 信号は一般的なモデムでは回線が接続されたときに ON となる。モデムによっては CD 信号と全く同じ挙動を示すものもある。DR 信号は 8251 のステータスレジスタ（I/O ポート 32H）の bit7 で調べることができる。List 7 にプログラムの例を示す。

List 7　DR 信号の状態取得

```
; RS-232C の DR 信号の状態を返す
; 戻り値は AX レジスタ： 0:DR=OFF, 1:DR=ON
        in      al,32h              ; bit5 が DR 信号の状態を表すビット
        rol     al,1
        and     ax,1
```

7 CI 信号の取得

CI 信号は主にモデムが着信したことを表すのに使用される。ただし、PC-9801 初代では CI 信号はハードウェア的に接続されていないので、ソフトウェアでは調べられない。CI 信号はシステムポート（I/O ポート 33H）の bit7 で調べられる。List 8 にプログラムの例を示す。

List 8　CI 信号の状態取得

```
; RS-232C の CI 信号の状態を返す
; 戻り値は AX レジスタ： 0:CI=OFF, 1:CI=ON
        in      al,33h              ; bit7 が CI 信号の状態を表すビット
        rol     al,1
        and     ax,1
```

ライブラリ

SetErOn

解説　RS-232C の ER 信号を ON にする。書式はつぎのとおり。

void **SetErOn**(void)

戻り値　なし

▶▶ CONTINUED

サンプル　RSSET.C

SetErOff

解説　RS-232C の ER 信号を OFF にする。書式はつぎのとおり。

void **SetErOff**(void)

戻り値　なし

サンプル　RSSET.C

SetRsOn

解説　RS-232C の RS 信号を ON にする。書式はつぎの通り。

void **SetRsOn**(void)

戻り値　なし

サンプル　RSSET.C

SetRsOff

解説　RS-232C の RS 信号を OFF にする。書式はつぎのとおり。

void **SetRsOff**(void)

戻り値　なし

サンプル　RSSET.C

CheckEr

解説　RS-232C の ER 信号の状態を返す。書式はつぎのとおり。

int **CheckEr**(void)

戻り値　RS-232C の ER 信号の状態。(0＝OFF、1＝ON)

サンプル　RSSTAT.C

CheckRs

解説　RS-232C の RS 信号の状態を返す。書式はつぎのとおり。

▶▶ CONTINUED

int **CheckRs**(void)

戻り値　RS-232CのRS信号の状態。(0=OFF、1=ON)

サンプル　RSSTAT.C

CheckCd

解説　RS-232CのCD信号の状態を返す。書式はつぎのとおり。

int **CheckCd**(void)

戻り値　RS-232CのCD信号の状態。(0=OFF、1=ON)

サンプル　RSSTAT.C

CheckCs

解説　RS-232CのCS信号の状態を返す。書式はつぎのとおり。

int **CheckCs**(void)

戻り値　RS-232CのCS信号の状態。(0=OFF、1=ON)

サンプル　RSSTAT.C

CheckCi

機能　RS-232CのCI信号の状態を返す。書式はつぎのとおり。

int **CheckCi**(void)

なお、初代PC-9801ではCI信号は調べられない。

戻り値　RS-232CのCI信号の状態。(0=OFF、1=ON)

サンプル　RSSTAT.C

CheckDr

機能　RS-232CのDR信号の状態を返す。書式はつぎのとおり。

int **CheckDr**(void)

戻り値　RS-232CのDR信号の状態。(0=OFF、1=ON)

サンプル　RSSTAT.C

シリアルポートのブレーク信号の送出

シリアルポートでは通常のデータの送受信の他に、ブレーク信号という特殊な信号を送信したり受信したりすることができる。

ブレーク信号とは、受信端子あるいは送信端子が長時間OFFになっている状態である。通常の調歩同期式でデータを送受信する場合は、1データ単位にかならずスタートビットとストップビットが入るため、送信端子や受信端子が長時間OFFに固定されることはない。そこで、十分に長時間OFFとなるブレーク信号を送れば、通信速度に関係なくブレーク信号を認識できる。

このため、ブレーク信号は通信速度の切り替えに利用したり、データ転送のキャンセル用などに利用することが多い。ここでは、そのブレーク信号の発生方法を解説する。

▶ポイント

- ブレーク信号は8251の送信端子を一定時間LOWレベルにすることで送信される。
- ブレーク信号の送信時間は75bps～9600bpsすべての通信速度でブレークと認識される時間(約300ミリ秒以上)に設定する。

▶プログラミングテクニック

1 ブレーク信号の送信を開始する

ブレーク信号を送出する場合は、8251のコマンドレジスタ(I/Oポート32H)のBREAKビット(bit3)を1にする。BREAKビットを1にすると、送信端子はLOWレベルになり、ブレーク信号が送信される(Figure 1参照)。

ただし、8251のコマンドレジスタは書き込んだ値を読み出せないので、一部のみの変更の際に過去の設定値が必要になる。ここでは、変数siocmdにコマンドレジスタへの設定値を保存させている。List 1にプログラム例を示す。

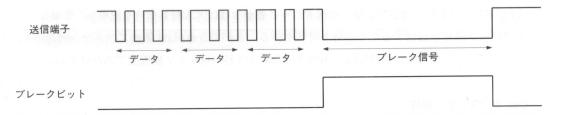
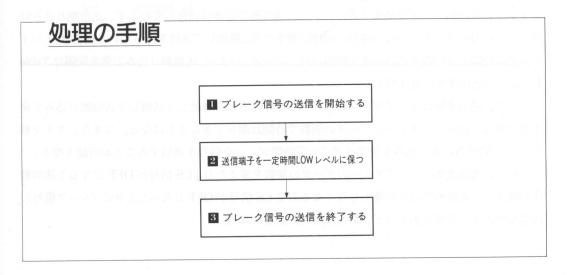



Figure 1　ブレーク信号送信時の動作

List 1　ブレーク信号の送信を開始する

```
; ブレーク信号の送信を開始する
        pushf
        cli
        mov     al,siocmd       ; 前回のコマンドレジスタへの出力値
        or      al,00001000b    ; BREAK ビットを 1 にする
        out     32h,al          ; 8251 のコマンドレジスタに出力
        mov     siocmd,al       ; 次回のために出力値を保存
        popf
```

2 送信端子を一定時間 LOW レベルに保つ

通信速度が 75bps 程度の低速なときでも十分にブレーク信号として判別できるようにするため、8251 のコマンドレジスタの BREAK ビットを 300 ミリ秒ほど 1 に保つ必要がある。この時間計測にはタイマ割り込みや VSYNC 割り込みなどを使用するのが一般的である。

また、ブレーク信号の送信中に他の処理を行う必要がなければ、OUT 命令を使用した簡単なループでブレーク信号を送信するための待ち時間を作ることができる。List 2 にプログラム例を示す。

なお、ブレーク信号の送信時間は、実用上は 200 ミリ秒〜600 ミリ秒程度であればよい。

List 2　ウェイト処理

```
; 200 ミリ秒〜600 ミリ秒程度のウェイト処理
        mov     cx,45000
wloop:
        out     5fh,al              ; ウェイトポート
        out     5fh,al
        out     5fh,al
        out     5fh,al
        out     5fh,al
        loop    wloop
```

I/O ポート 5FH はウェイトポートと呼ばれており、PC-H98 シリーズで用意されたものだが、他機種でも同様に使用できる。このポートになんらかのデータを OUT 命令で出力すると、CPU に一定時間（0.6μ 秒〜2μ 秒程度）ウェイトが入るようになっている。

ブレーク信号送信中も他の処理を行う必要があるのであれば、このようなウェイト処理は適さず、タイマ割り込みや VSYNC 割り込みなどを使用したほうがよい。ただし、PC-9800 シリーズでは、タイマ割り込みは 1 つのプログラムでしか利用できない貴重な資源である。また、VSYNC 割り込みは CRT BIOS と競合し、他の常駐プログラムなどとの相性が問題となることが多い。

ブレーク信号のための時間計測には、送信割り込みを利用する方法もある。ブレーク信号の送信を開始すると 8251 の送信端子は LOW になる。このとき、データを送信しても送信端子は LOW のままなので送信データは消失する。しかし、送信動作自体は通常と変わらず、送信割り込みはブレーク信号を送信していない場合と同様に発生する。連続して送信を行っている場合は、1 バイトの送信時間と送信割り込みの発生間隔は同じになる。つまり、送信割り込みの発生間隔は Table 1 のように通信速度に反比例する。

そこで、通信速度によってダミーのデータ（00H など）を連続して送信して送信割り込みを発生させれば、送信したダミーのデータの個数で時間計測ができることになる。つまり、タイマ割り込みや VSYNC 割り込みを使用せずに一定時間ブレーク信号を送信することが可能となる。

しかし、通信速度によってダミーのデータの個数を変えたり、CS 信号が OFF になると送信動作が停止して送信割り込みが発生しなくなるので CS 信号が OFF になったときにブレーク信号の送信を中止する必要があるなど、繁雑な処理が必要になる。

Table 1　通信速度と送信割り込み発生間隔（データ長8、ストップビット1、パリティなし）

通信速度（bps）	間隔（ミリ秒）
75	133.33
150	66.67
300	33.33
600	16.67
1200	8.33
2400	4.17
4800	2.08
9600	1.04
19200	0.52
38400	0.26

3 ブレーク信号の送信を終了する

ブレーク信号の送信を終了するには、8251のコマンドレジスタのBREAKビットを0にすればよい。List 3にプログラム例を示す。

List 3　ブレーク信号の送信を終了する

```
; ブレーク信号の送信を終了する
        pushf
        cli
        mov     al,siocmd       ; 前回のコマンドレジスタへの出力値
        and     al,11110111b    ; BREAKビットを0にする
        out     32h,al
        mov     siocmd,al
        popf
```

▶ワンポイント

■ブレーク信号を受信した場合の動作

ブレーク信号を受信したとき、8251は複数の00Hを受信した場合と似た動作をする。しかし、調歩同期式では、データは1つのスタートビットとストップビットに挟まれて転送される。したがって、1つのデータの長さ以上にLOWな状態が続くと、スタートビットとストップビットがないことになり、フレーミングエラーが発生しフレーミングエラーフラグがONになる。さらに長くLOWレベルの信号が続くと、ブレーク信号と認識してブレーク状態フラグがONになる(Figure 2参照)。

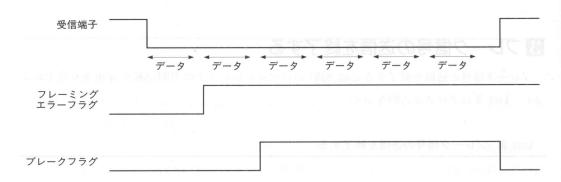



Figure 2　ブレーク信号受信時の動作

8251のステータスのフレーミングエラーフラグはフレーミングエラーが発生すると1になり、エラークリアのコマンドを8251に発行すると0になる。エラークリアをするには、8251のコマンドレジスタ(I/Oポート32H)のbit4を1にする。エラークリアを行わない場合は、フレーミングエラーフラグは1のままである。

エラーフラグの状態は8251の送受信動作にはまったく影響しないので、エラー状態を監視する必要がなければエラーフラグをクリアする必要はない。ブレーク信号を受信したかどうかを認識しなければならないことはあまり多くないので、添付のライブラリではブレーク状態のフラグのチェックは省略している。

ライブラリ

RsSendBreak

解説 約 300 ミリ秒のブレーク信号を送信する。書式はつぎのとおり。

void **RsSendBreak**(void)

本ライブラリ中の RsOpen 関数で初期化していなければならない。

戻り値 なし

RsBreakOn

解説 ブレーク信号の送信を開始する。書式はつぎのとおり。

void **RsBreakOn**(void)

本ライブラリ中の RsOpen 関数で初期化していなければならない。

戻り値 なし

RsBreakOff

解説 ブレーク信号の送信を終了する。書式はつぎのとおり。

void **RsBreakOff**(void)

本ライブラリ中の RsOpen 関数で初期化していなければならない。

戻り値 なし

シリアルポートの通信速度の取得

シリアルポートの通信速度はインターバルタイマの8253に分周値として設定するが、その設定値は直接8253から読み出すことができない。また、RS-232C BIOSも初期化時の通信速度をメモリに記録していない。

シリアルポートの操作が単一のプログラム内で完結している場合は、プログラムで通信速度を設定する際にその値をメモリに保存しておけば現在の通信速度は簡単に取得できる。しかし、通信ソフトの子プロセスから起動した、まったく別のプログラムや、特定のホットキーで起動する常駐プログラムなどからは、シリアルポートの通信速度を簡単には取得できない。多くの通信ソフトではRS-232C BIOSを使用せずに直接8253を操作して通信速度を設定しているので、BIOSを監視するといった方法もとれない。

そこで、8253のカウンタ値から8253の設定値を推定し通信速度を取得する方法を解説する。

▶ポイント

- 通信速度の設定は8253にクロックの分周値を設定して行われている。
- 通信速度の設定値自体を8253から取得する方法はない。
- 8253のカウンタはカウントダウンを繰り返している。
- 8253から設定値は読み出せないが、カウントダウン中のカウンタは読み出せる。
- カウンタを繰り返し読み出して、読み出した値から通信速度を推測する。
- カウンタの値は必ず設定値以下となるので、カウンタの値の上限を調べれば、設定された値から通信速度がわかる。

▶プログラミングテクニック

1 カウンタの値を読み出す

8253のカウンタは、設定された分周値を初期値として、クロック信号によってカウントダウンを続けている。カウント値が0になると、自動的にカウンタに初期値が設定され、繰り返しカウントダウンが行われる。

8253には設定した分周値を読み出す機能はないが、現在カウント中のカウンタの値を読み出す機能がある。これは8253にラッチコマンド(80H)を発行して行う。List 1にプログラム例を示す。

List 1　カウンタの読み出し

```
; 結果:DX レジスタ
        pushf
        cli
        mov     al,80h          ; タイマのカウンタ#2へのラッチコマンド
        out     77h,al          ; コマンドを発行
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        in      al,75h          ; カウント値の下位バイトを読み出す
        mov     dl,al
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        in      al,75h          ; カウント値の上位バイトを読み出す
        mov     dh,al
        popf
```

さて、8253のカウンタは約2MHzあるいは約2.5MHzの速度でカウントダウンを行っている。このカウントは高速なため、カウントダウンの様子をプログラムで詳細に読み取るのは不可能である。

しかし、Figure 1(A)のようにカウンタへの設定値が十分に大きいときはカウントダウンの繰り返し周期が長いため、Figure 1(B)のようにカウントダウンの様子が推測できる値が読み取れる。

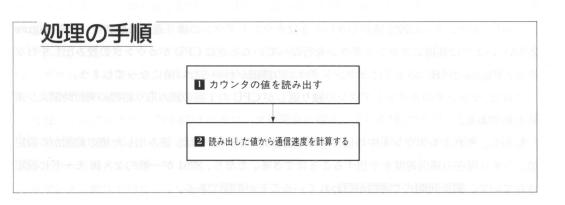
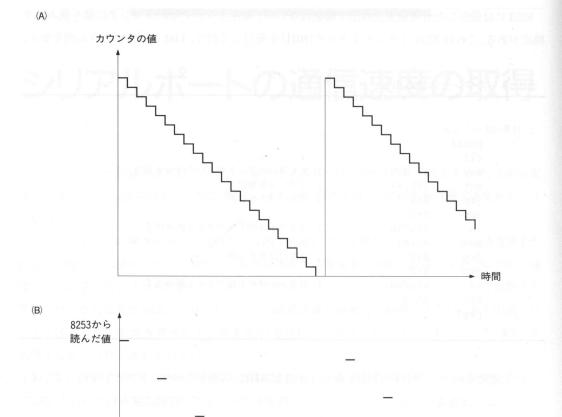



Figure 1　設定値が大きい場合のカウントダウンの読み取り

しかし、カウンタへの設定値が小さいときはカウントダウンの繰り返し周期が短いため、Figure 2（A）のように高速にカウントダウンを行なっているときにCPUがカウンタの読み出しを行なうと、Figure 2（B）のようにカウントダウンの詳細がわからない値になってしまう。

これは、カウンタのカウントダウンの繰り返しがCPUの1回の読み取り処理の実行時間より速いためである。

しかし、それでもカウント中の値を十分な回数だけ読み出せば、読み出した値の範囲から設定値、つまり現在の通信速度を予想することはできる。ただし、8251が一般的な×16モードに設定されていて、調歩同期式で通信が行われていることが前提である。

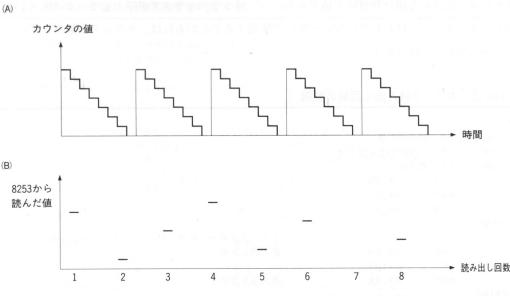



Figure 2　設定値が小さい場合のカウントダウンの読み取り

シリアルポートの通信速度は、75、150、300、1200、2400、…というように、75を基準とした2のべき乗の値になっている。8253の設定値は通信速度に反比例しているので、結局8253の設定値も2のべき乗になっている。このように通信速度が不連続となっているので、8253のカウンタから読み出した値が設定値の半分より大きな値であれば、設定値より小さい値でも通信速度が判定できる。

カウンタから読み出した値は0から設定値までの範囲内であるから、仮にカウンタから読み出した値が設定値の半分以下となる確率が1/2と仮定する。そうすると、カウンタから単純に20回読み出しを行った場合でも、読み出し結果の最大値が設定値の半分以下となる確率、つまり、通信速度の取得に失敗する確率は約100万分の1($1 \div 2^{20}$)になる。

しかし、カウントダウンの繰り返し周期がCPUの読み取り速度より大幅に長い場合があるので、実際はカウンタから読み出した値すべてが設定値の半分以下となることもある。そこで、必ず8253のカウントダウンの繰り返し周期よりCPUの読み取り処理の繰り返しが長くなるように、読み取り回数を増やす工夫が必要となる。

そこで、ここではカウンタからの読み取り回数を20回にするのではなく、List 2のようにカウントダウンの1周期分を、つまり読み取った値が前の値より大きくなるまでを調べ、これが20回になるまで繰り返すようにする。

なお、このプログラムでは8253のカウンタが完全に停止している場合(カウンタの読み出し結果が常に同じ値の場合)でも20回でループが終了するように、1回前の読み出し結果と同じ値が読み出された場合も、ループカウンタを1つ減らしている。

カウンタの動作が停止している場合はシリアルポートの通信速度は0(bps)であるが、カウン

タから読み出される値は無意味な値である。したがって、カウンタが停止している状態（シリアルポートがまったく使われていない状態）で使用することがあれば、カウンタが動作しているかどうかも判定したほうがよい。

List 2　カウンタ読み出し回数の制御

```
; AX レジスタ：読み出し値
; DX レジスタ：1 回前の読み出し値
; BX レジスタ：最大値
        mov     cx,20           ; ループカウンタを設定
        mov     dx,0
        mov     bx,0
loop1:
        …                       ; カウンタ読み出し処理：AX レジスタに読み出し値
        cmp     ax,bx           ; 最大値と比較
        jb      skip1
        mov     bx,ax           ; 最大値を更新
skip1:
        cmp     ax,dx           ; 1 回前と比較
        jb      skip2
        dec     cx              ; ループカウンタを減らす
skip2:
        mov     dx,ax           ; 次回のために記録
        or      cx,cx
        jnz     loop1           ; 終わりでなかったら繰り返し
```

2 読み出した値から通信速度を計算する

カウンタの設定値の近似値が取得できれば、List 3 のようなプログラムで通信速度を推測することができる。カウンタへの設定値はシステムクロックが 5MHz 系か 8MHz 系かによって異なる。

List 3　カウンタ値から通信速度を判定する

```
; call DX レジスタ：カウンタの設定値の近似値
; ret  AX レジスタ：判定通信速度 (0〜9)
;            0   1   2   3   4    5    6    7    8     9
; 通信速度  75 150 300 600 1200 2400 4800 9600 19200 38400 (bps)
        sub     ax,ax               ; システム共通域のセグメント：0000h
        mov     es,ax
        mov     dx,2048             ; 5MHz 系の基底分周値
        mov     al,es:[0501h]       ; システム共領域の BIOS-FLAG
        test    al,80h              ; システムクロックが 5MHz 系か 8MHz 系かのチェック
        jz      count               ; 速度判定処理へ
        mov     dx,1664             ; 8MHz 系の基底分周値
count:
        sub     ax,ax               ; 速度判定値（戻り値）を基底の 75bps にセット
calc:
        shr     dx,1                ; 基底分周値を 1/2 にする
        cmp     dx,bx               ; カウント最大値と比較
        jb      pend                ; 基底分周値がカウント最大値より小なら終了へ
        inc     ax                  ; 速度判定値（戻り値）を次の値にする
        jmp     SHORT calc
pend:
```

ライブラリ

GetSpeed

解説　シリアルポートの現在の通信速度を取得し、通信速度に応じた数値を返す。書式はつぎのとおり。

int **GetSpeed**(void)

戻り値　通信速度と戻り値の関係は次のようになる。

戻り値	0	1	2	3	4	5	6	7	8	9
通信速度	75	150	300	600	1200	2400	4800	9600	19200	38400

サンプル　RSSTAT.C

簡単な通信プログラムの作成

本章では RS-232C 制御のためのライブラリを作成したが、その使用例として簡単な通信ソフトを作成する。アップロードやダウンロード機能すらないシンプルなプログラムであるが、プログラムの根本的な構造は大部分の通信ソフトと同じである。信号線の制御などの機能は利用していないが、送受信割り込みをはじめ多くの基幹のとなる関数を利用している。

本ライブラリ関数を使って、通信プログラムを作成する際の参考としてもらいたい。

▶ポイント

- 全2重通信を実現するために、キー入力のチェックと受信文字のチェックを交互に行う。
- キー入力のチェックと受信文字のチェックの頻度が同じだと表示が遅くなるので、受信文字のチェックの頻度を上げる。

▶プログラミングテクニック

1 シリアルポートを初期化する

プログラムを簡単にするため、通信パラメータはプログラム中では、9600bps、8 ビットパリティなし、RS/CS フロー制御ありに固定している。シリアルポートのハードウェアを直接操作しているため、MS-DOS に RSDRV.SYS などの RS-232C 用デバイスドライバを組み込んでおく必要はない。実際の初期化には、RsOpen 関数を使う。

2 キーボードからの入力を確認する

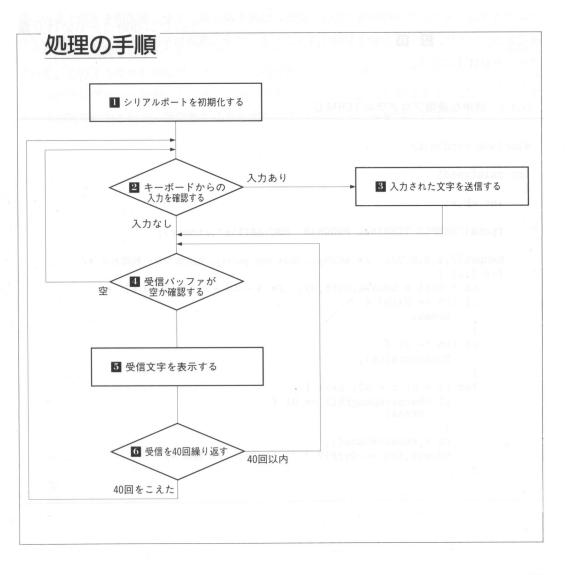
キーボードからの入力には C 言語のライブラリの bdos 関数を用いて、MS-DOS のダイレクトコンソール入出力ファンクションを呼び出してリアルタイムでキー入力を行っている。kbhit 関数と getch 関数の組み合わせでは SHIFT ＋ STOP でキーバッファクリアを行った場合に次のキー入力まで処理が戻ってこないとう不都合がある。

ESC キーが入力されたらプログラムを終了する。また、入力がなければ、4 へ飛びシリアルポートからの受信を行う。

3 入力された文字を送信する

TransData関数により文字の送信を行う。サンプルプログラムでは、キー入力の速度より送信速度が速いことを期待して、送信バッファの空きチェックを省略している。ただし、ファイルからのアップロードなどの処理では、送信バッファの空きチェックは必須である。

4 受信バッファが空か確認する

ReceiveLength関数で受信したデータがあるかどうかを確認する。受信バッファが空であれば、2へ飛び、次のループを実行する。

5 受信文字を表示する

受信バッファが空でなければ、ReceiveData 関数により受信したデータを受け取り、bdos 関数で MS-DOS のダイレクトコンソール入出力ファンクションを呼び出して画面に文字を表示する。

6 受信を 40 回繰り返す

本プログラムでは画面表示を高速化するために、キー入力のチェック 1 回につき、受信のチェックと表示を最大 40 回行うようにしている。通信では一般的に、キー入力より受信（画面表示）のほうがデータ量が多いので、受信処理に重点を置く必要があるためである。40 回受信を繰り返すと 2 に戻りキーの入力を受け付ける。

全二重通信の場合は送信と受信を並行して処理しなければならないが、MS-DOS のようなシングルタスクのシステムでは短時間で送信と受信の処理を繰り返して全二重通信を実現するのが普通である。つまり、2～6 の動作を繰り返すことよって全二重通信を実現している。実際のプログラムを List 1 に示す。

List 1　簡単な通信プログラム TERM.C

```
#include <stdio.h>

int main(void)
{
    int ch,n;

    fputs("SAMPLE TERMINAL PROGRAM. ESC:EXIT\n",stderr);

    RsOpen(7,1,2,0,2);  /* 9600bps, 8bit non parity, RS/CS フロー制御あり */
    for (;;) {
        ch = 0xff & bdos(6,0xff,0);  /* キースキャン (ノーウェイト) */
        if (ch == 0x1b) {
            break;
        }
        if (ch != 0) {
            TransData(ch);
        }
        for (n = 0; n < 40; n++) {
            if (ReceiveLength() == 0) {
                break;
            }
            ch = ReceiveData();
            bdos(6,(ch == 0xff)? ' ': ch,0);  /* 表示 */
        }
```

```
    }
    RsClose();

    fputs("\nEXIT\n",stderr);
    return (0);
}
```

▶ワンポイント

■使わなかった機能について

プログラムを簡単にするため、送信バッファがいっぱいになったかどうかのチェックは省略している。このようなサンプルプログラムでは送信割り込みを使用しているメリットはないが、ファイルのアップロード処理など、多くのデータを送信する場合は送信動作を止めずにディスクアクセスが可能になるなど、送信割り込みを使用しているメリットが発揮される。

割り込み編

ハードウェア割り込みの基礎知識

インターバルタイマやシリアルポート、マウスなどを直接制御するプログラムでは、かならずハードウェア割り込み機能を使用することになる。ハードウェア割り込みをコントロールしている周辺LSIは、8259という割り込みコントローラ(PIC)である。

ハードウェア割り込み機能を活用するためには、ハードウェア割り込みに関係するハードウェアとソフトウェアを理解する必要がある。

ハードウェア割り込みの概要

CPUの割り込み機能にはソフトウェア割り込みとハードウェア割り込みと例外処理がある。ハードウェア割り込みは周辺LSIなどからのハードウェア的な要因により、CPUにあらかじめ決められた処理を行わせるための機能である。ハードウェア割り込みの処理は、ソフトウェア割り込みの処理と同じようにIRET命令で終了する。

PC-9800シリーズでは、キーボード、インターバルタイマ、シリアルポート、マウス、フロッピーディスク、ハードディスクなど、多くの周辺装置がハードウェア割り込み機能を利用している。複数のハードウェア割り込みをうまく管理しているのが、割り込みコントローラである。

PC-9801には2個の割り込みコントローラが搭載されており、一方をマスタ割り込みコントローラ、もう一方をスレーブ割り込みコントローラと呼んでいる。割り込みコントローラには、割り込み要求信号を受けつける端子と、CPUに割り込み要求を出す端子が付いており、PC-9800シリーズではFigure 1のような構成になっている。

なお、ハードウェア割り込みにはNMI（Non Maskable Interrupt）という割り込みもあるが、一般的にはアプリケーションソフトが利用する割り込みではないため、本編ではNMIについての解説は省略する。

ハードディスクインターフェイスなど、ボード上のディップスイッチにより割り込み信号が変更できるものもある。ハイレゾモードも割り込みコントローラ周辺についてはノーマルモードと基本的には同じである。ただし、マウスの割り込みはノーマルモードでは通常は割り込み信号として拡張スロットINT6を使用するが、ハイレゾモードではINT2に固定されている。

割り込みコントローラは1個につき8種類のハードウェア割り込みしか扱えないため、PC-9801では2個の割り込みコントローラを使用し、ディジーチェイン（数珠つなぎ）接続してサポート

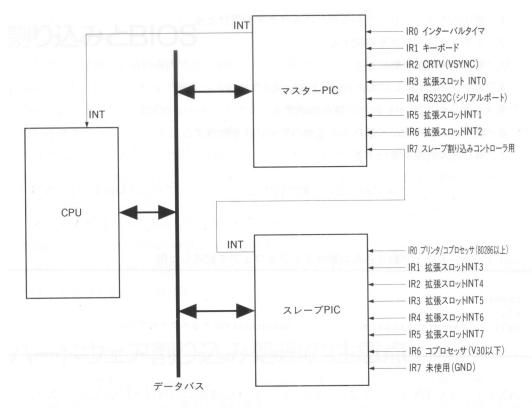



Figure 1　割り込みコントローラの構成

できるハードウェア割り込みの数を増やしている。

このため、マスタ割り込みコントローラ側に接続されたハードウェア割り込み処理と、スレーブ割り込みコントローラ側に接続されたハードウェア割り込み処理では、割り込みコントローラに対する制御方法が異なる。

PC-9801ではマスタ割り込みコントローラはI/Oポート00HとI/Oポート02Hに、スレーブ割り込みコントローラはI/Oポート08HとI/Oポート0AHに割り当てられている。詳細は『テクニカルデータブック』の「割り込みコントローラ」のI/Oポートと命令の項を参照していただきたい。

ハードウェア割り込み発生時の動作

ハードウェア割り込みが発生すると実行中のプログラムは中断され、代わりに割り込み処理が実行される。ハードウェア割り込みが発生したとき、CPUはソフトウェア的には自動的につぎのような動作をする。80386以上のCPUで仮想86モードを利用している場合はCPUはもっと複雑な動作をするが、アプリケーションプログラムから見たCPUの動作は同じと考えてよい。

1. 割り込みコントローラから割り込みベクタを受けとる
2. スタックにフラグを格納する
3. 割り込み禁止状態になる
4. スタックにCSレジスタの値を格納する
5. スタックにIPレジスタの値を格納する
6. 割り込みベクタから割り込み処理のアドレスを取得する
7. 割り込み処理を実行する

したがって、ハードウェア割り込みの動作自体は、ソフトウェア的にはList 1のようなプログラムの動作と似ている。

List 1　ハードウェア割り込みの動作をソフトウェアで記述した例

```
pushf
cli
call    far ptr xxxxxxxx           ; xxxxxxxxは対応するベクタのアドレス
```

AXレジスタやBXレジスタなど、通常のレジスタは自動的には保存されない。そこで、ハードウェア割り込み処理では、最初に必ず使用するレジスタを保存する必要がある。

ハードウェア割り込み処理はメインのプログラムの実行をハードウェア的なタイミングで中断して実行される。したがって、割り込み処理の作成には十分な注意が必要である。また、実行できる内容も限られる。ハードウェア割り込み処理の作り方が悪いとメインプログラムに悪影響を与えたり、システムが正常に動作しなくなったりしてしまう。

ハードウェア割り込み処理では、ソフトウェア割り込み処理と同じように処理の終了をIRET命令で行う。IRET命令は、つぎのような動作をする。

1. スタックからIPレジスタの値を復帰する
2. スタックからCSレジスタの値を復帰する
3. スタックからフラグを復帰する

こうして、CPUはハードウェア割り込み発生直前に実行していた処理を再開する。

割り込みとBIOS

ハードウェア割り込み処理内では、原則としてMS-DOSのシステムコールは使えない。また、BIOSルーチンもハードウェア割り込み処理で使用できるようには作られていないので、原則として使用できない。また、BIOSルーチンのエントリ部分でSTI命令を実行しているものも使用できないと考えていい。たとえば、タイマ割り込みの中でカレンダ時計の読み出しのBIOSルーチンを呼び出してはならない。このBIOSルーチンは再入対策が施されていないからである。

なんらかの再入対策がされているBIOSルーチンや、再入対策が不要なほど簡単な処理しかしていないBIOSルーチンはハードウェア割り込み中でも利用できる。それでも、BIOSルーチンのエントリ部でSTI命令を実行してしまうものは条件によっては使用できない。

BIOSルーチンは処理時間や割り込み禁止時間、再入対策、ハードウェア割り込み中で使用可能かどうかなどが外部仕様に明記されていないので、一般的にはハードウェア割り込み処理中でBIOSルーチンを使用するのは危険である。

ハードウェア割り込み処理の注意点

ハードウェア割り込みは、他のプログラムの処理中に実行されるという点に常に注意して作成する必要がある。ハードウェア割り込みが発生したときのフラグレジスタの内容も予測不可能である。そこで、ハードウェア割り込み処理中にMOVSB命令やSTOSB命令など、方向フラグに影響される命令を使用する場合は、かならずCLD命令あるいはSTD命令で方向フラグを設定してから使用する。

ハードウェア割り込みが発生した時点では、スタックエリアがどこにあるか(OS内か、アプリケーション内か)が不明なため、十分なスタックエリアがあることは保証されない。したがって、ハードウェア割り込み処理がスタックを比較的多く使用する場合は、List 2のプログラムのように早急にローカルスタックに切り替えを行う必要がある。ただし、ローカルスタックを使用した割り込み処理では、再入禁止の対策が必要である。

ハードウェア割り込み処理は、再入防止やリアルタイム性向上のために、割り込み禁止状態のまま実行することが多い。ただし、そうすると割り込み処理の実行中は他の割り込みが禁止されてしまう。

CLI命令を実行してハードウェア割り込みを禁止してから、STI命令あるいはPOPF命令などを実行して割り込みを受けつける状態になるまでの間の時間を、割り込み禁止時間という。PC-9801では、キーボードやシリアルポートなど、多くの周辺装置の制御に割り込みを使用しているので、割り込み禁止時間を長くしすぎるとさまざまな問題が生じる。

PC-9801ではキーボードインターフェイスが常に19200bpsで、シリアルポートでは製品仕様として最大9600bpsまで使用され、他にはタイマ割り込みなどが使用される。シリアルポートを19200bpsや38400bpsで使用している場合は、許容される割り込み禁止時間はさらに短くなる。さらに、送受信割り込みを使用していれば、割り込み発生の頻度はさらに上がる。

これらの割り込み処理の実行時間と、割り込み禁止時間の合計がハードウェア割り込みの発生時間より長いと、それぞれの割り込み処理が間に合わなくなり、シリアルポートの受信文字の取りこぼしなどの問題が発生してしまう。

List 2　ローカルスタックの切り換え

```
intr     proc far
         push     ax
         mov      cs:save_ss,ss
         mov      cs:save_sp,sp
         mov      ax,cs
         mov      ss,ax
         mov      sp,offset cs:my_stack
          :
          :                               ; レジスタ待避
          :
          :                               ; 割り込み処理
          :
          :                               ; 割り込み終了処理 (EOI 発行)
          :
          :                               ; レジスタ復帰
         mov      ss,cs:save_ss
         mov      sp,cs:save_sp
         pop      ax
         iret

         even
save_ss dw        0
save_sp dw        0

         dw       100 dup (0)
my_stack label word
intr     endp
```

これらを処理するCPUの速度も問題になる。ハードディスクへのアクセス中などにもハードウェア割り込みは発生する。このようなデータ転送のDMAが発生しているときは、CPUの見かけ上の実行速度は大幅に低下する。また、80386以上のCPUで仮想86モードを利用していると、特定のI/Oポートの入出力命令などでI/Oトラップが発生する。この場合、通常は1μ秒程度で実行できるI/O命令でも、実行時間が見かけ上30μ秒程度になることがあるので注意が必要である。

このように、実行速度の遅い機種や仮想86モードでも問題を起さないようにするためには、割り込み処理の実行時間や割り込み禁止時間は極力短くする必要がある。

目安としては、ハードウェア割り込み処理などのように割り込み禁止状態で実行するプログラ

ムは50命令以下になるようにするとよいだろう。また、割り込み禁止状態でrep movsb命令などでデータ転送を行いたい場合は、かならず128バイト以内、できれば64バイト以内にする。

こうしたハードウェア割り込みへの配慮は、割り込み処理側のプログラムだけでなく、アプリケーションプログラムを作成する側にも必要である。たとえば、周辺LSIへのアクセスがハードウェア割り込みにより分断されないように配慮するとか、ハードウェア割り込みのために十分にスタックエリアを用意しておくとか、スタックポインタの操作を慎重に行うなどの配慮は当然ながら必要である。

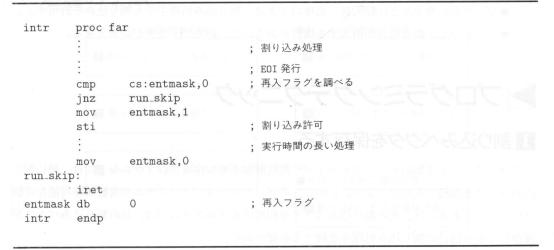
再入チェックの方法

PC-9800シリーズでは、COPYキーやSTOPキーが押されると、最初にキーボードのハードウェア割り込みが発生する。このハードウェア割り込み処理中でINT 05HやINT 06Hが実行され、画面コピーなどの処理が行われる。

画面コピーなど、実行時間が非常に長くなる処理をハードウェア割り込みを使用して起動する場合は、ハードウェア割り込み処理の中でEOIを発行してからSTI命令で割り込みを許可し、必要とする処理を行うのが一般的である。このようにすれば、たとえば画面コピー処理の間でも他の割り込みが止まることは防げる。

このような場合は、割り込み処理が完全に終わらないうちに同じハードウェア割り込みが発生することが考えられる。そのため、List 3のプログラムのように再入チェック処理を設ける必要がある。再入チェック処理をしないとスタックオーバーフローなどによりシステムが暴走することがある。

List 3　再入チェックのプログラム

```
intr     proc far
         :
         :                                ; 割り込み処理
         :
         :                                ; EOI 発行
         cmp      cs:entmask,0            ; 再入フラグを調べる
         jnz      run_skip
         mov      entmask,1
         sti                              ; 割り込み許可
         :
         :                                ; 実行時間の長い処理
         mov      entmask,0
run_skip:
         iret
entmask db        0                       ; 再入フラグ
intr     endp
```

割り込みベクタのフック

ハードウェア割り込みをソフトウェアで監視する最も簡単な方法は、割り込みベクタを書き換え、本来の割り込み処理ではなく自前の割り込み処理が起動されるようにすることである。これを「割り込みベクタのフック」と呼ぶ。割り込みベクタが変更されなければ、この方法で簡単にハードウェア割り込みを監視することができる。

割り込みベクタをフックしても、特定の条件では本来の割り込み処理を呼ぶようにしたり、通常は本来の割り込み処理を使っておき特定の条件で特別な動作をさせたりすることも考えられる。

ここでは、割り込みベクタをフックする方法を、タイマ割り込みとキーボードの割り込みを中心に解説し、実践的な例としてキークリック音を出すプログラムを考える。

▶ポイント

- 割り込み処理は、登録、割り込み発生時の処理、プログラム終了時の処理の3つに大きくわかれる
- 割り込みをフックするときは、従来のベクタも保存し自分の割り込み処理の中でそれを呼び出すようにすべきである。
- プログラムを終了するときは、保存した元々のベクタを復旧するべきである。
- ビープ音の発生など比較的長い処理のときは、割り込み処理中でも割り込みを許可する。
- ハードウェア割り込みが再入する場合があるので、再入チェックが必要となる。

▶プログラミングテクニック

1 割り込みベクタを保存する

ブロックデバイス型のデバイスドライバや常駐解除不能な常駐プログラムなどでは、単に割り込みベクタを登録するだけでよい。しかし、アプリケーションプログラムや常駐解除可能な常駐プログラム、あるいは、元の割り込みベクタを利用するプログラムでは、元の割り込みベクタを保存してから自己の割り込み処理を登録する必要がある。

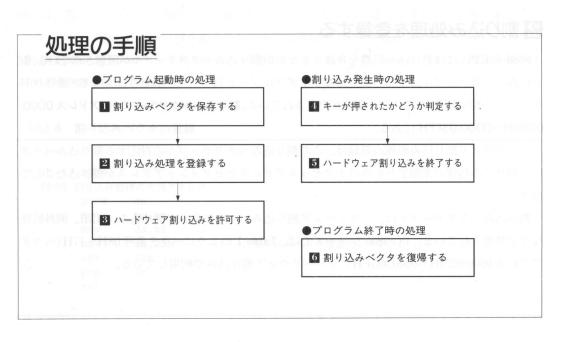
たとえば、ベクタ番号09Hのキーボード割り込みをフックする場合は、List 1のようにする。

List 1　キーボード割り込みベクタの保存

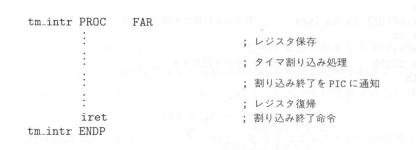
```
        mov     ax,3508h            ; INT 08H のベクタを読み出す
        int     21h
        mov     iv08ofs,bx
        mov     iv08seg,es          ; ベクタを保存する

        push    ds
        mov     dx,OFFSET cs:tm_intr    ; 割り込み処理の先頭アドレス
        mov     ax,cs
        mov     ds,ax
        mov     ax,2508h            ; INT-08H のベクタを設定する
        int     21h
        pop     ds

         :                          ; タイマの設定
         :                          ; タイマ割り込み許可
         :                          ; メインプログラム
         :                          ; タイマ割り込み禁止

        push    ds
        lds     dx,iv08adr
        mov     ax,2508h            ; INT 08H のベクタを元に戻す
        int     21h
        pop     ds
         :                          ; プログラム終了処理
```

なお、上記のプログラムでは、同一のコードセグメント内で割り込み処理がList 2のように定義されていることを想定している。また、DSレジスタが示すデータセグメントにList 3のような割り込みベクタ保存用のワークエリアがあることを想定している。

List 2　割り込み処理の記述

```
tm_intr PROC    FAR
         :                        ; レジスタ保存
         :                        ; タイマ割り込み処理
         :                        ; 割り込み終了をPICに通知
         :                        ; レジスタ復帰
        iret                      ; 割り込み終了命令
tm_intr ENDP
```

List 3　割り込みベクタの保存領域

```
iv09adr label   dword
iv09ofs dw      0                 ; オフセット用
iv09seg dw      0                 ; セグメント用
```

2 割り込み処理を登録する

8086系CPUには割り込み処理を登録するための割り込みベクタテーブルが用意されている。割り込みベクタテーブルは1つの割り込みで1ダブルワード(4バイト)を使用し、ベクタ番号00Hからベクタ番号FFHまでの256個分が用意されている。**割り込みベクタテーブルはアドレス0000:0000H〜0000:03FFH**にある。

ハードウェア割り込み処理の登録は、この割り込みベクタテーブルの対応する割り込みベクタに、割り込み処理の先頭アドレスのオフセットアドレスとセグメントアドレスを書き込むことで行う。

割り込みベクタテーブルは、ソフトウェア割り込み、ハードウェア割り込み、NMI、例外処理などで共用されている。PC-9800シリーズでは、Table 1のようにベクタ番号08H〜17H(ベクタアドレス0000:0020H〜0000:005FH)をハードウェア割り込みで利用している。

Table 1　ハードウェア割り込み一覧

ベクタ番号	ベクタアドレス	用　途
マスタ割り込みコントローラ		
08H	0000:0020H	インターバルタイマ
09H	0000:0024H	キーボード
0AH	0000:0028H	CRTV (VSYNC)
0BH	0000:002CH	拡張バス INT0
0CH	0000:0030H	RS-232C
0DH	0000:0034H	拡張バス INT1
0EH	0000:0038H	拡張バス INT2
0FH	0000:003CH	システム予約 (不完全割り込み)
スレーブ割り込みコントローラ		
10H	0000:0040H	プリンタ/コプロセッサ (80286 以上)
11H	0000:0044H	拡張バス INT3 (ハードディスク)
12H	0000:0048H	拡張バス INT41 (640K バイトフロッピーディスク)
13H	0000:004CH	拡張バス INT42 (1M バイトフロッピーディスク)
14H	0000:0050H	拡張バス INT5 (サウンド)
15H	0000:0054H	拡張バス INT6 (マウス)
16H	0000:0058H	コプロセッサ (V30 以下)
17H	0000:005CH	システム予約 (不完全割り込み)

ハードウェア割り込みの利用状況は機種や周辺機器の接続状況によって異なる。ハードウェア割り込みの多くは、本体に内蔵している周辺装置に割り当てられている。

新たに拡張スロットに、ハードウェア割り込みを使用するインターフェイスボードを装着する場合は、他のハードウェア割り込みと競合しないように割り込み番号を設定する必要がある。

ハードウェア割り込みの処理を割り込みベクタに登録するには、List 4 のプログラムのようにMS-DOSのシステムコールを利用するのが簡単である。あるいは、List 5 のように直接、割り込みベクタを書き換えてもよい。

List 4　割り込みベクタの登録

```
; AL ベクタ番号
; ES:DX 割り込み処理の先頭アドレス
        push    ds
        mov     bx,es
        mov     ds,bx
        mov     ah,25h              ; 割り込みベクタ設定のファンクション
        int     21h
        pop     ds
        ret
```

List 5　直接割り込みベクタの書き換え

```
; 割り込み処理の登録
; AL ベクタ番号
; ES:DX 割り込み処理の先頭アドレス
        pushf
        cli
        push    ds
        sub     ah,ah
        shl     ax,1            ; ベクタ番号を4倍してベクタのアドレスを計算
        shl     ax,1
        mov     bx,ax
        sub     ax,ax           ; 割り込みベクタのセグメント:0000H
        mov     ds,ax
        mov     [bx],dx         ; オフセットアドレスを書き込む
        mov     [bx+2],es       ; ベクタアドレスを書き込む
        pop     ds
        popf
        ret
```

割り込み処理内でベクタを書き換える場合は、MS-DOSのシステムコールは利用すべきではない。割り込み処理内で直接にベクタを書き換えれば、実行時間や再入の可否、フラグレジスタなどに関して不確定要素が入る余地はない。

3 ハードウェア割り込みを許可する

ハードウェア割り込みを利用するためには、割り込みベクタの登録のほかに、割り込みコントローラへの設定も必要となる。

CLI命令とSTI命令を使用すればハードウェア割り込みの禁止と許可が行える。しかし、これでは特定のハードウェア割り込みのみの制御ができない。特定のハードウェア割り込みのみを禁止したり許可したりしたい場合は、割り込みコントローラのIMR（割り込みマスクレジスタ）を操作する。

IMRは、割り込みコントローラに割り当てられているI/Oポートを読むだけで取得することができる。これは、マスタ割り込みコントローラではI/Oポート02H、スレーブ割り込みコントローラではI/Oポート0AHである。

マスタ割り込みコントローラのIMRの各ビットは、実際のハードウェア割り込みとFigure 1のように対応している。

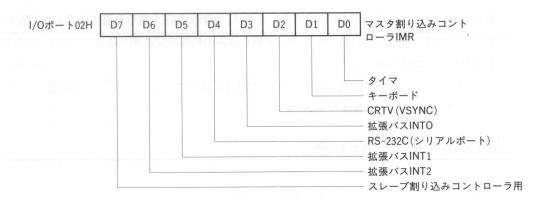



Figure 1　割り込みコントローラの IMR の意味

各ビットを 1 にすれば割り込みの禁止に、0 にすれば割り込みの許可になる。この IMR を操作すれば、上記のハードウェア割り込みを自由に禁止や許可をすることができる。したがって、たとえばタイマ割り込みの禁止や許可は List 6 や List 7 のようなプログラムで実現できる。

List 6　タイマ割り込みの禁止

```
; タイマ割り込み禁止
tm_di proc
        pushf
        cli
        in      al,02h          ; マスタ割り込みコントローラの IMR を読む
        or      al,00000001b    ; タイマ割り込み禁止
        out     02h,al
        popf
        ret
tm_di endp
```

List 7　タイマ割り込みの許可

```
; タイマ割り込み許可
tm_ei proc
        pushf
        cli
        in      al,02h          ; マスタ割り込みコントローラの IMR を読む
        and     al,11111110b    ; タイマ割り込み許可
        out     02h,al
        popf
        ret
tm_ei endp
```

割り込みコントローラのIMRを操作するときは、かならずCLI命令を実行してすべて割り込みを禁止した状態で行う。これは、ハードウェア割り込みの処理の間に割り込みコントローラのIMRが変更される可能性があるからである。

なお、割り込み許可をSTI命令を使用せずにPOPF命令で行っているのは、List 8のようなサブルーチンの利用方法への配慮である。

List 8　stiを使わない方がよい例

```
cli
call    tm_di           ; タイマ割り込みを禁止
:
:                       ; 割り込み禁止状態のまま行う作業
call    tm_ei           ; タイマ割り込み許可
sti
```

POPF命令の代わりにSTI命令を使用してしまうと、このような方法は使えなくなる。このようなサブルーチン形式で使用することがなければ、CLI命令とSTI命令の組み合わせでもよい。

キーボードのように常にハードウェア割り込みが許可されていることが想定できる場合は、ハードウェア割り込みの許可の処理を省略してもかまわない。

4 キーが押されたかどうか判定する

キーボードからのハードウェア割り込みは、キーを押したときと離したときに発生する。そこで、キーが押されたときのみクリック音を出すためには、キーを押したときかどうかの判定が必要になる。この判定には、キーボードの受信データを監視する方法と、キーコードバッファの格納状態の変化を調べる方法が考えられる。

まず、キーコードバッファの状態を監視する方法である。キーが押されると、キーボード用の割り込み処理によってキーコードバッファが更新される。このとき、システム共通域にあるワークエリアも値が変化する（Table 2参照）。したがって、本来のキーボード割り込み処理が実行される前後で、このワークエリアが変化したかどうかを調べればよい。

Table 2　システム共通域

アドレス	意　味
0000:0528H	キーコードバッファ内のキーコードの数（KB_COUNT）

プログラム例をList 9に示す。割り込みベクタの保存や設定の処理はすでに説明しているので省略する。

List 9　キーが押されたかどうかの判定

```
iv_enter proc far
        push    ax
        push    es
        sub     ax,ax               ; システム共通域:セグメント 0000H
        mov     es,ax
        mov     al,es:[0528h]       ; キーコードバッファ内のキーコード数
        mov     cs:key_n,al
        pop     es
        pop     ax

        pushf
        call    dword ptr cs:iv_adr        ; 本来のキーボード割り込み処理

        push    ax
        push    es
        sub     ax,ax               ; システム共通域:セグメント 0000H
        mov     es,ax
        mov     al,es:[0528h]       ; キーコードバッファ内のキーコード数
        cmp     al,cs:key_n
        pop     es
        pop     ax
        jz      iv_end

        cmp     cs:entmask,0        ; 再入チェック
        jnz     iv_end
        mov     cs:entmask,1
        sti                         ; 割り込みを許可
        push    ax
        mov     al,06h              ; ビープ ON
        out     37h,al
        mov     ax,1000
cloop:
        out     5fh,al              ; ウェイトポート
        dec     ax
        jnz     cloop
        mov     al,07h              ; ビープ OFF
        out     37h,al
        pop     ax
        mov     cs:entmask,0
iv_end:
        iret
iv_enter endp

iv_adr  label   dword
iv_ofs  dw      0                   ; INT 09H オフセット保存用
iv_seg  dw      0                   ; INT 09H セグメント保存用

entmask db      0                   ; 再入禁止フラグ
key_n   db      0                   ; キーコード数のワークエリア
```

キーコードバッファの状態を監視する方法では、CTRL キーや SHIFT キーなど、キーバッファが更新されないキーの押下ではクリック音が出ない。しかし、キーボードからの受信データそのものを調べれば、すべてのキーの押下でクリック音を出すことができる。

キーボードからの受信データは**I/Oポートの41H**で読むことができ、本来のキーボード割り込み処理が終了しても、つぎにキーボードからつぎのデータが来ない限り、同じ値を保っている。

そこで、本来のキーボード割り込み処理の後にこれを読み出し、キーが押されたのか、離されたのかを受信データのMake/Breakビットを調べて判定する（Figure 2参照）。なお、このI/Oポートは本来のキーボード割り込み処理を実行する前に読み出してはいけない。本来のキーボード割り込み処理の実行前にキーデータを読み出してしまうと、I/Oポート43HにあるステータスレジスタのRxRDYビットがクリアされてしまうからである。

Figure 2　受信データのMakeとBreakの判定

プログラム例をList 10に示す。割り込みベクタの保存や設定の処理はすでに説明しているので省略する。

List 10　キーボードが押されかたどうかの判定

```
iv_enter proc far
        pushf
        call    dword ptr cs:iv_adr     ; 本来のキーボード割り込み処理

        push    ax
        in      al,43h          ; ステータス
        test    al,02h          ; 受信データがあるか
        pop     ax
        jnz     iv_end          ; 受信データがあれば受信データは読まない
        push    ax
        in      al,41h          ; キーボードからの受信データを読む
        test    al,80h          ; 0:Make, 1:Break
        pop     ax
        jnz     iv_end

        cmp     cs:entmask,0    ; 再入チェック
        jnz     iv_end
```

```
 mov     cs:entmask,1
        sti                             ; 割り込みを許可
        push    ax
        mov     al,06h                  ; ビープ ON
        out     37h,al
        mov     ax,1000
cloop:
        out     5fh,al                  ; ウェイトポート
        dec     ax
        jnz     cloop
        mov     al,07h                  ; ビープ OFF
        out     37h,al
        pop     ax
        mov     cs:entmask,0
iv_end:
        iret
iv_enter endp

iv_adr  label   dword
iv_ofs  dw      0                       ; INT 09H オフセット保存用
iv_seg  dw      0                       ; INT 09H セグメント保存用

entmask db      0                       ; 再入禁止フラグ
```

5 ハードウェア割り込みを終了する

ハードウェア割り込み処理では、割り込み処理の終了をかならず割り込みコントローラに通知しなければならない。割り込み処理の終了の通知は、割り込みコントローラのOCW2（オペレーションコマンドワード）レジスタにEOI（End Of Interrupt）コマンドを出力することで行う。EOIコマンドは20Hである（Figure 3参照）。

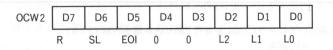



Figure 3　OCW2 コマンドのフォーマット

EOIコマンドを発行するプログラムは、ハードウェア割り込みがマスタ割り込みコントローラで制御されているものか、スレーブ割り込みコントローラで制御されているものかによって異なる。

OCW2はマスタ割り込みコントローラではI/Oポートの00Hに、スレーブ割り込みコントローラでは08Hに割り当てられている。マスタ割り込みコントローラにEOIを発行するプログラムを

List 11　割り込み処理の終了（マスタ割り込みコントローラの場合）

```
; マスタ割り込みコントローラに EOI を発行
          mov       al,20h              ; EOI コマンド
          out       00h,al              ; PIC に EOI を発行
```

マスタ割り込みコントローラで制御されているハードウェア割り込みは、このように簡単だが、スレーブ割り込みコントローラで制御されているハードウェア割り込みでは、割り込みコントローラがディジーチェインになっているため、少し複雑になる。

スレーブ割り込みコントローラに割り当てられているハードウェア割り込みの割り込み終了処理では、最初にスレーブ割り込みコントローラに EOI を発行する。つぎにスレーブ割り込みコントローラの ISR（インサービスレジスタ）を読み出し、他にハードウェア割り込みが受け付けられているかどうかを調べる。ISR レジスタは、割り込みコントローラに ISR リードコマンドを発行してから I/O ポート 08H から値を読み出す（Figure 4 参照）。

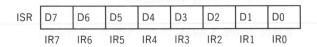



Figure 4　ISR コマンドのフォーマット

ISR は割り込みが受けつけられているビットが 1 になっているので、全ビットが 0 であればほかに割り込みは受けつけられないことになる。ISR の全ビットが 0 の場合は、マスタ割り込みコントローラにも EOI を発行する。EOI の発行は割り込み禁止状態で行う（List 12 参照）。

List 12　割り込み処理の終了（スレーブコントローラの場合）

```
; スレーブ割り込みコントローラに EOI を発行（割り込み禁止状態で行う）
          mov       al,20h
          out       08h,al              ; スレーブ PIC に EOI を発行
          mov       al,0bh
          out       08h,al              ; OCW3 に ISR リードコマンドを発行
          jmp       $+2                 ; リカバリータイム用
          in        al,08h              ; ISR を読み出す
          or        al,al
          jnz       skip                ; 割り込みがあればマスタ PIC に EOI を発行しない
          mov       al,20h
          out       00h,al              ; マスタ PIC に EOI を発行
skip:
```

6 割り込みベクタを復帰する

プログラムの終了時などでは、最初に保存した割り込みベクタをふたたび復帰する。それには、最初に変数に保存した割り込みベクタを、ふたたび割り込み処理の登録と同じ方法で登録する。

COLUMN

仮想86モードでの割り込み

EMSドライバなどで80386や80486の仮想86モードを利用していると、ハードウェア割り込みやソフトウェア割り込みの動作は非常に複雑になる。

仮想86モードを管理しているプログラムのことを仮想86モニタと呼ぶ。この仮想86モニタは、80386や80486の機能を利用して割り込みベクタテーブルを0000:0000H以外の場所（通常はプロテクトメモリ上）に移してしまう。

割り込みが発生すると、仮想86モニタの割り込みベクタテーブルに登録されている、仮想86モニタの割り込みシミュレーション処理が実行される。この割り込みシミュレーション処理は、仮想86モニタが関知しない割り込みであれば、0000:0000Hから始まる本来の割り込みベクタテーブルに設定された割り込み処理を呼び出す。ここで、ようやく通常の割り込み処理が起動されるのである。

仮想86モードでは、ソフトウェア割り込みもハードウェア割り込みもこのように仮想86モニタを通して処理される。このため、割り込み処理の実行時間は、仮想86モードを使用しない場合と比較して30μ秒～100μ秒ほど増加する。

割り込みコントローラを再初期化して割り込みベクタ番号を変更すると、割り込みコントローラは新しく設定されたベクタ番号でCPUに割り込みを要求する。このとき、仮想86モニタによるI/Oポートのエミュレーション処理が不完全で、割り込みコントローラが再初期化された場合に対応していないと、システムが正しく動かない。

割り込みコントローラの再初期化

割り込み処理のフックでは、割り込みベクタの書き換えによる方法以外に、割り込みコントローラの再初期化による方法がある。独自に割り込みコントローラを再初期化すれば、ハードウェア割り込みの8個の割り込みベクタを一括して別の割り込みベクタ番号に割り当てることができる。ただし、これは最終手段であり、日常的に利用するような方法ではないことに注意する必要がある。

ここでは例として、割り込みコントローラを再初期化してハードウェア割り込みを変更し、フックと同じキークリック音を出すプログラムを考えてみる。

▶ ポイント

- ハードウェア割り込みのベクタ番号は割り込みコントローラに設定する。
- 割り込みコントローラを再初期化すれば割り込みベクタの変更が可能となる。
- ハードウェア割り込み中から本来の割り込み処理を呼び出す。

▶ プログラミングテクニック

1 割り込みコントローラを再初期化する

ハードウェア割り込みのベクタ番号は、マスタ割り込みコントローラではベクタ番号08H〜0FHに、スレーブ割り込みコントローラではベクタ番号10H〜17Hに割り当てられている。このベクタ番号はハードウェア的に固定されているのではなく、割り込みコントローラの初期化時にソフトウェアで設定しているものである。

割り込みコントローラにはICW1、ICW2、ICW3、ICW4というレジスタがあり、このICW2でベクタ番号を指定することができる(Figure 1参照)。詳細については、『テクニカルデータブック』の「割り込みコントローラ」のイニシャライズコマンドワード(ICW)の項を参照していただきたい。

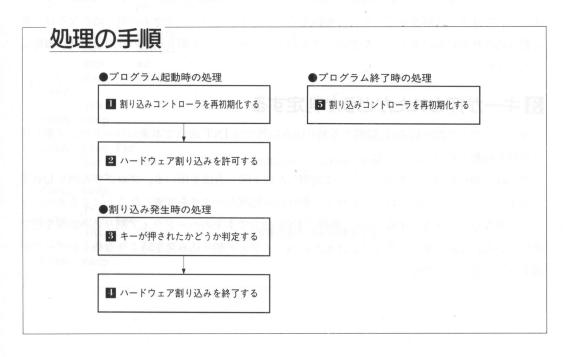
ICW2のbit 0〜bit 2は、ベクタ番号の設定とは無関係のビットなので、0を設定する。

Figure 1　ICW コマンドのフォーマット

割り込みコントローラのICW2のみの設定を変更することはできないため、ICW2を変更する場合は割り込みコントローラの再初期化を行う必要がある。プログラム例を List 1 に示す。割り込みコントローラの再初期化は割り込み禁止状態で行う必要がある。

新しく割り当てられたベクタには、それぞれ本来のベクタを呼び出すような割り込み処理を前もって登録しておく。

キーボードのハードウェア割り込みはマスタ割り込みコントローラに割り当てられているので、マスタ割り込みコントローラを再初期化する。この場合、スレーブ割り込みコントローラを再初期化する必要はない。

割り込みコントローラはシステムの割り込み処理の中核であるので、通常のアプリケーションプログラムが割り込みコントローラを再初期化することはない。そこで、これを逆手にとり、割り込みコントローラを再初期化して割り込みベクタ番号を変更すれば、ほぼ完全にハードウェア割り込みを監視することが可能となる。再初期化の際には、現在の割り込みコントローラの設定が変更されないように注意する必要がある。

List 1　割り込みコントローラの再初期化

```
; マスタ割り込みコントローラの再初期化（標準設定に戻す）
        mov     al,11h
        out     00h,al          ; ICW1 を書き込む
        mov     al,08h          ; 使用ベクタ番号 08H～0FH
        out     02h,al          ; ICW2 を書き込む
        mov     al,80h
        out     02h,al          ; ICW3 を書き込む
        mov     al,1dh
        out     02h,al          ; ICW4 を書き込む

; スレーブ割り込みコントローラの再初期化（標準設定に戻す）
        mov     al,11h
        out     08h,al          ; ICW1 を書き込む
        mov     al,10h          ; 使用ベクタ番号 10H～17H
        out     0ah,al          ; ICW2 を書き込む
        mov     al,07h
        out     0ah,al          ; ICW3 を書き込む
        mov     al,09h
        out     0ah,al          ; ICW4 を書き込む
```

なお、複数のプログラムがこの方法を使用した場合は、当然ながら最後に再初期化したときの設定が有効となる。

2 ハードウェア割り込みを許可する

ハードウェア割り込みの許可は割り込みコントローラのIMRを操作して行う。IMRの変更方法についてはすでに解説しているので省略する。キーボードのハードウェア割り込みならば、常に割り込み許可状態にあることが想定できるので、ハードウェア割り込みの許可の処理は省略してもかまわない。

3 キーが押されたかどうか判定する

各ハードウェア割り込みは、監視する割り込み以外ではINT命令で本来のハードウェア割り込み処理を起動するようにしておく。

判定は「割り込みベクタのフック」で説明したのと同じ方法を用いる。プログラム例をList 2に示す。このプログラムでは、ハードウェア割り込み処理のベクタを変更したことによるオーバーヘッドをなるべく少なくするために、単純にINT命令で本来のハードウェア割り込み処理を呼び出している。このため、スタックには元々のハードウェア割り込み発生時よりも多くのデータが積まれていることになる。

List 2　キーが押されたかどうかの判定

```
iv_08h  proc far
        int     08h             ; 本来の割り込み処理を起動
        iret
iv_08h  endp

iv_09h  proc far
        int     09h             ; 本来のキーボード割り込み処理を起動

        push    ax
        in      al,43h          ; ステータス
        test    al,02h          ; 受信データがあるか
        pop     ax
        jnz     iv_end          ; 受信データがあれば受信データは読まない
        push    ax
        in      al,41h          ; キーボードからの受信データを読む
        test    al,80h          ; 0：Make、1：Break
        pop     ax
        jnz     iv_end

        cmp     cs:entmask,0    ; 再入チェック
        jnz     iv_end
        mov     cs:entmask,1
        sti                     ; 割り込みを許可
        push    ax
        mov     al,06h          ; ビープ ON
        out     37h,al
        mov     ax,1000
cloop:
        out     5fh,al          ; ウェイトポート
        dec     ax
        jnz     cloop
        mov     al,07h          ; ビープ OFF
        out     37h,al
        pop     ax
        mov     cs:entmask,0
iv_end:
        iret
iv_09h  endp

iv_0ah  proc far
        int     0ah             ; 本来の割り込み処理を起動
        iret
iv_0ah  endp

iv_0bh  proc far
        int     0bh             ; 本来の割り込み処理を起動
        iret
iv_0bh  endp
```

```
iv_0ch  proc far
        int     0ch              ; 本来の割り込み処理を起動
        iret
iv_0ch  endp

iv_0dh  proc far
        int     0dh              ; 本来の割り込み処理を起動
        iret
iv_0dh  endp

iv_0eh  proc far
        int     0eh              ; 本来の割り込み処理を起動
        iret
iv_0eh  endp

iv_0fh  proc far
        int     0fh              ; 本来の割り込み処理を起動
        iret
iv_0fh  endp

entmask db      0                ; 再入禁止フラグ
```

スタックの状態を通常のハードウェア割り込み発生時と同じにしたい場合は、割り込み処理をList 3のプログラムのようにINT命令を使用しない方法に変える必要がある。実際に割り込みベクタを読み出しているので、割り込みベクタが変更された場合にも対処できる。

なお、この方法ではハードウェア割り込みの応答性がINT命令を使用した場合より多少悪くなるが、この程度のオーバーヘッドで実際に問題が発生することはまずない。

List 3　INT命令を利用しない割り込み処理

```
iv_08h  proc far
        push    ax
        push    ds
        sub     ax,ax                    ; 割り込みベクタ：セグメント0000H
        mov     ds,ax
        lds     ax,ds:[08h*4]            ; INT 08Hの割り込みベクタを読み出す
        mov     word ptr cs:now_08h,ax
        mov     word ptr cs:now_08h+2,ds ; コードセグメントに待避
        pop     ds
        pop     ax
        jmp     dword ptr cs:now_08h     ; 本来の割り込み処理を起動

now_08h dd      0                        ; far jmp用
iv_08h  endp
```

4 ハードウェア割り込みを終了する

元々のハードウェア割り込み処理の中で、割り込みコントローラに対してEOIを発行しているので、とくに何も処理をする必要がなければIRET命令で終了するだけでよい。

5 割り込みコントローラを再初期化する

割り込みベクタを標準設定に戻すには、標準設定のベクタ番号を設定して割り込みコントローラを再初期化する。プログラム例をList 4に示す。

List 4　割り込みコントローラを元の状態に初期化

```
; マスタ割り込みコントローラの再初期化（割り込みベクタを 88H～8FH に設定）
        mov     al,11h
        out     00h,al          ; ICW1 を書き込む
        mov     al,88h          ; 使用ベクタ番号 88H～8FH
        out     02h,al          ; ICW2 を書き込む
        mov     al,80h
        out     02h,al          ; ICW3 を書き込む
        mov     al,1dh
        out     02h,al          ; ICW4 を書き込む

; スレーブ割り込みコントローラの再初期化（割り込みベクタを 90H～97H に設定）
        mov     al,11h
        out     08h,al          ; ICW1 を書き込む
        mov     al,90h          ; 使用ベクタ番号 90H～17H
        out     0ah,al          ; ICW2 を書き込む
        mov     al,07h
        out     0ah,al          ; ICW3 を書き込む
        mov     al,09h
        out     0ah,al          ; ICW4 を書き込む
```

タイマ割り込み

タイマ割り込みはINT 1CHのBIOSでインターバルタイマ機能としてサポートされている。ハイレゾモードでは複数の処理を行うマルチイベントや、一定間隔で繰り返し実行するリピートモード、タイマキャンセル機能をサポートしている。しかし、ノーマルモードでは単純なインターバルタイマの機能しかない。

ここでは、ノーマルモードやハイレゾモードにかかわらず自由にタイマ割り込みを占有できる方法、つまりタイマBIOSを使わずに直接ハードウェア割り込みを制御する方法を解説する。ただし、タイマを他の常駐プログラムなどと同時にマルチイベントで使用したい場合は、この方法は適さない。

▶ポイント

- インターバルタイマはカウンタ#0を使用する。
- カウンタ#0はI/Oポート77Hで動作モードの設定などを行う。
- カウンタ#0はI/Oポート71Hでカウンタ値の設定などを行う。
- システムクロックが5MHz系と8MHz系では設定するカウンタ値が異なる。
- タイマ割り込みは割り込みベクタ番号08Hを使用する。
- タイマ割り込み許可や禁止は割り込みコントローラのIMRを操作して行う。

▶プログラミングテクニック

1 タイマ割り込みの利用状況を調べる

PC-9800シリーズでは8253という型番の周辺LSIを搭載している。8253はカウンタ#0、カウンタ#1、カウンタ#2という3個のカウンタを持っており、インターバルタイマやシリアルポートの通信速度の設定、ビープ音の音階の作成などに使用している。

8253やインターバルタイマ全般についての説明は、『テクニカルデータブック』の「タイマ」を参照のこと。

タイマ割り込みを占有して使いたい場合、すでにタイマ割り込みが使われていたらプログラム

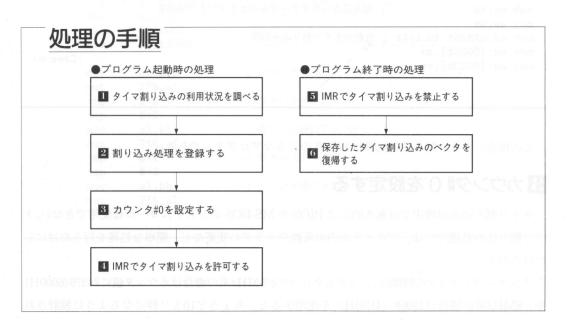
の実行を中止するか、タイマ割り込みを強制的に横取りするようにする。すでにタイマ割り込みが使用されているかどうかは、割り込みコントローラのIMRを調べれば判断できる。プログラムをList 1に示す。

List 1　タイマ割り込みの利用状況の調査

```
; タイマ割り込みが使用されているかどうかを調べる
; 結果 AL 00H：使用中 01H：未使用
        in      al,02h          ; 割り込みコントローラのIMRを読む
        and     al,00000001b
```

ただし、この方法はタイマ割り込みがほかのプログラムで使用されていて、一時的に割り込み禁止になっている場合を考慮していない。

2 割り込み処理を登録する

割り込みベクタの設定や保存については、すでに説明したとおりである。タイマ割り込みの保存処理のプログラム例をList 2に示す。

List 2　タイマ割り込みの保存処理

```
pushf
cli
sub     ax,ax             ; 割り込みベクタテーブルのセグメント：0000H
mov     es,ax
mov     ax,es:[0020h]     ; タイマ割り込み INT 08H
mov     dx,es:[0022h]
mov     iv_ofs,ax
mov     iv_seg,dx
```

このプログラムでは、List 3のようなワークエリアがあることを前提としている。

List 3　必要なワークエリア

```
iv_ofs  dw      0               ; オフセットアドレス保存用
iv_seg  dw      0               ; セグメントアドレス保存用
```

引き続き、List 4のように自前のタイマ割り込み処理を割り込みベクタに登録する。

List 4　タイマ割り込みベクタに登録

```
sub ax,ax                  ; 割り込みベクタテーブルのセグメント：0000H
mov es,ax
mov ax,offset tm_intr ; 自前のタイマ割り込み処理
mov es:[0020h],ax
mov es:[0022h],cs
```

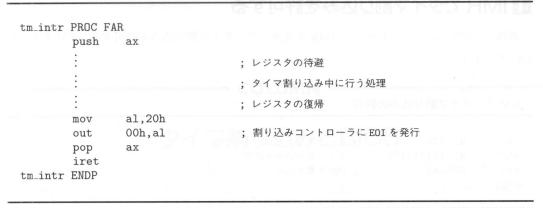
この場合、タイマ割り込み処理はList 5のようなプログラムである。

3 カウンタ#0を設定する

タイマ割り込み処理中では基本的にはBIOSやMS-DOSのシステムコールは使用できない。タイマ割り込み処理内では、プログラム内の変数やフラグの変更など、簡単な処理を行うだけにしたほうがよい。

インターバルタイマの時間は、システムクロックが5MHz系の場合はカウンタ値に24576(6000H)を、8MHz系の場合は19968（4E00H）を使用すると、ちょうど10ミリ秒になるように設計され

List 5　想定される割り込み処理

```
tm_intr PROC FAR
        push    ax
        :                               ; レジスタの待避
        :                               ; タイマ割り込み中に行う処理
        :                               ; レジスタの復帰
        mov     al,20h
        out     00h,al                  ; 割り込みコントローラに EOI を発行
        pop     ax
        iret
tm_intr ENDP
```

ている。倍の値にすれば 20 ミリ秒単位にできるが、10 ミリ秒にした方が時間計測などに便利である。

インターバルタイマは、実際には 10 ミリ秒で繰り返し割り込みを発生させる必要があるので、カウンタ#0 の設定はモード 3（方形波レートジェネレータ）を使用する。プログラム例を List 6 に示す。

List 6　インターバルタイマの設定

```
        sub     ax,ax           ; システム共通域のセグメント:0000H
        mov     es,ax
        mov     bx,24576        ; システムクロックが 5MHz 系の場合のカウンタ値
        mov     al,es:[0501h]   ; システム共通域の BIOS-FLAG
        test    al,80h          ; システムクロックのチェック
        jz      tm_set2
        mov     bx,19968        ; システムクロックが 8MHz 系の場合のカウンタ値
tm_set2:
        mov     al,36h          ; カウンタ#0 をモード 3 で使用する
        out     77h,al
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        mov     al,bl           ; カウンタ値の下位バイト
        out     71h,al
        jmp     $+2             ; リカバリタイム用
        jmp     $+2
        mov     al,bh           ; カウンタ値の上位バイト
        out     71h,al
```

4 IMR でタイマ割り込みを許可する

最後に、割り込みコントローラの IMR を変更して、タイマ割り込みを許可する。プログラムを List 7 に示す。

List 7　タイマ割り込みの許可

```
in      al,02h          ; 割り込みコントローラの IMR
and     al,11111110b    ; タイマ割り込みを許可
out     02h,al          ; IMR を書き込む
popf
```

5 IMR でタイマ割り込みを禁止する

アプリケーションプログラムを終了する場合には、タイマ割り込みの使用を中止するために、List 8 のようにして割り込みを禁止する。

List 8　タイマ割り込みの禁止

```
pushf
cli
in      al,02h          ; 割り込みコントローラの IMR を読み出す
or      al,00000001b    ; タイマ割り込みを禁止
out     02h,al          ; IMR を書き込む
```

6 保存したタイマ割り込みのベクタを復帰する

最後に、List 9 のようにして割り込みベクタを元に戻す。利用中のタイマ割り込みを強制的に横取りした場合は、ベクタを元に戻した後にタイマ割り込みを許可したほうがよい。

List 9　保存したベクタの復帰

```
sub     ax,ax               ; セグメント：0000H
mov     es,ax
mov     ax,iv_ofs
mov     dx,iv_seg
mov     es:[0020h],ax       ; ベクタを書き戻す
mov     es:[0022h],dx
popf
```

COLUMN

タイマ割り込みとBIOS

PC-9800シリーズには、一定間隔で発生するハードウェア割り込みには、タイマ割り込み、CRTV割り込み、マウス割り込みがある。正確な時間間隔で割り込みを発生させたい場合、マウス割り込みはマウスドライバが使用するので使えず、CRTV割り込みは動作モードなどによって発生間隔が異なり、CRTV割り込みを使用するBIOSもあるので、完全に定期的な割り込みは期待できない。つまり、PC-9800シリーズでは、正確な時間間隔で割り込みを発生させる場合は、タイマ割り込みを使用するしかない。

ところが、タイマ割り込みはBIOSによるサポートがとても弱い。BIOSからはタイマ割り込みはインターバルタイマとしてしか使えず、いわゆるリピートモードはサポートされていない。また、途中でのキャンセルもできず、BIOSに設定した値も読み出せない。

このようにBIOSに必要な機能がない場合、要求される仕様を満足させるためにハードウェアを直接アクセスしなければならないことになる。システムがあらかじめ一定の周期でタイマ割り込みを使用していれば、それに合わせてタイマ割り込みのベクタをフックして動作させるという方法が使えるのだが、PC-9801の場合システムがタイマ割り込みをまったく使用していないので、そのような手法も使えない。

システム時計

PC-9800シリーズでは、システム時計にμPD1990あるいはμPD4990というICを使用している。このICは、時刻の読み出しなどをシリアルで行わなければならないので、BIOSを使用して時刻の読み出しを行うと、1ミリ秒～2ミリ秒程度の実行時間が必要となる。

実行時間が長くなるため、時計読み出しのBIOSは、その中でハードウェア割り込みを許可している。しかし、このBIOSは再入に対応していない。そのためハードウェア割り込み中などで現在時刻を知りたい場合でも、BIOSを呼び出してシステム時計を読み出すようなことはしてはならない。時刻の読み出しは時間がかかるので、ハードウェア割り込み中で直接時計用のICをアクセスする方法も使えない。

現在では、時計用のICはRAMと同じようにデータを簡単に読み出せるものが主流であり、PC-9800シリーズでは時計用のICも弱点となっている。

CRTV 割り込み

CRTV (VSYNC) 割り込みは CRT コントローラの垂直同期信号によって発生する割り込みである。そのため、1 秒間に数十回定期的に発生する。しかし、垂直同期信号といってもノーマルモードは約 56Hz、ハイレゾモードは約 80Hz、PC-H98 シリーズはもう 1 つ異なる周波数があり、PC-98LT や PC-98HA もこれらとは異なる周波数であり、機種ごとにその回数は違うと考えた方がよい。

CRTV 割り込みが発生した瞬間は、必ずディスプレイのブランク区間であるため、このタイミングで画面の切り替えやスクロールを行うと、乱れのない切り替えやスクロールが可能になる。

ここでは CRTV 割り込みの利用法について解説する。

▶ ポイント

- I/O ポート 64H に任意の値を書き込むと 1 回だけ CRTV 割り込みが発生する。
- これを CRTV 割り込み処理内で行えば、繰り返し CRTV 割り込みが発生する。
- BIOS などの中には CRTV 割り込みを禁止するものがある。これらの BIOS の実行後は、適宜 CRTV 割り込みを許可する。

▶ プログラミングテクニック

1 本来の CRTV 割り込みのベクタを保存する

タイマ割り込みと同様に本来の CRTV の割り込みベクタを保存する。プログラムを List 1 に示す。

List 1　CRTV 割り込みベクタの保存

```
mov     ax,350ah
int     21h                 ; INT-0AH ベクタ読み出し
mov     iv_ofs,bx
mov     iv_seg,es
```

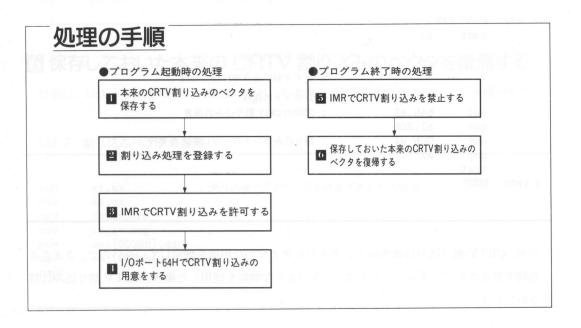
2 割り込み処理を登録する

割り込み処理の登録もタイマ割り込みなどの場合と同様である。割り込みベクタの書き換えは、割り込み禁止状態で実行する。プログラムを List 2 に示す。

List 2　割り込み登録

```
pushf
cli
sub     ax,ax                  ; 割り込みベクタテーブルのセグメント:0000H
mov     es,ax
mov     ax,offset v_intr       ; 自前の CRTV 割り込み処理
mov     es:[0028h],ax
mov     es:[002ah],cs
```

3 IMR で CRTV 割り込みを許可する

CRTV 割り込みの許可は、タイマ割り込みと同様に割り込みコントローラの IMR を操作すればよい。この処理は割り込み禁止状態で実行する。プログラムを List 3 に示す。

List 3　CRTV 割り込みの許可

```
in      al,02h          ; 割り込みコントローラの IMR
and     al,11111011b    ; CRTV 割り込みを許可
out     02h,al          ; IMR を書き込む
popf
```

4 I/O ポート 64H で CRTV 割り込みの用意をする

割り込みコントローラの IMR で CRTV 割り込みを許可しただけでは CRTV 割り込みは発生しない。I/O ポート 64H に任意の値を出力して、CRTV 割り込みのための用意をする必要がある。プログラムを List 4 に示す。I/O ポート 64H への出力が複数回になるのはかまわない。

List 4　CRTV 割り込みの用意

```
out     64h,al
```

CRTV 割り込みをインターバルタイマのように使用したい場合は、List 5 のように割り込み処理の中で I/O ポート 64H に何か値を出力し、次回の CRTV 割り込みを用意させる。

List 5　連続して CRTV 割り込みを発生させるためのプログラム

```
v_intr  PROC FAR
        push    ax
        :                       ; レジスタの待避
        :                       ; タイマ割り込み中に行う処理
        :                       ; レジスタの復帰
        out     64h,al          ; 次回の CRTV 割り込みの用意
        mov     al,20h
        out     00h,al          ; 割り込みコントローラに EOI を発行
        pop     ax
        iret
v_intr  ENDP
```

なお、CRTV 割り込みは本来はインターバルタイマ的な用途を想定していないので、さまざまな処理で禁止されてしまう。たとえば、つぎのような機能を使用した場合、CRTV 割り込みは禁止されてしまう。

- MS-DOS エスケープシーケンス 画面表示行数の変更
- CRT BIOS CRT モードの設定
- CRT BIOS 1 つの表示領域の設定
- CRT BIOS カーソルタイプの設定
- CRT BIOS フォントパターンの読み出し
- CRT BIOS ユーザ文字の定義

そこで、このような機能を使用した場合は、再度 CRTV 割り込みを許可し、I/O ポート 64H に任意の値を出力して CRTV 割り込みが発生するようにする。メインプログラムで定期的に CRTV 割り込みを許可させる方法も考えられる。

常駐プログラムなどで CRTV 割り込みを使用する場合は、CRT BIOS をフックして、本来の CRT BIOS の実行後に CRTV 割り込みを許可するように細工する。

5 IMR で CRTV 割り込みを禁止する

CRTV 割り込みの禁止は、タイマ割り込みと同様に割り込みコントローラの IMR を操作すればよい。プログラムを List 6 に示す。

List 6　CRTV 割り込みの禁止

```
pushf
cli
in      al,02h          ; 割り込みコントローラの IMR
or      al,00000100b    ; タイマ割り込みを禁止
out     02h,al          ; IMR を書き込む
```

6 保存しておいた本来の CRTV 割り込みのベクタを復帰する

最後に、List 7 のようにして、保存しておいた本来の CRTV 割り込みのベクタを復帰する。

List 7　割り込みベクタの復帰

```
sub     ax,ax           ; 割り込みベクタテーブルのセグメント:0000H
mov     es,ax
mov     dx,iv_ofs
mov     cx,iv_seg
mov     es:[0028h],dx
mov     es:[002ah],cx
popf
```

その他周辺機器編

その他周辺機器編

ドライブ名のDA/UAへの変換

DISK BIOSでディスクアクセスするときは、MS-DOSのドライブ名をそのまま使うことはできない。DISK BIOSではDA/UA（Device Address/Unit Address）という値を使ってディスクドライブを指定する。DA/UAは、装置の種類と同種の装置の中の装置番号から成る値である。

したがって、DISK BIOSを利用する際には、MS-DOSのドライブ名をDA/UAに変換する必要がでてくる。ここでは、その方法について解説する。

▶ポイント

- MS-DOS 3.1最終版（製品名PS98-011）以降のMS-DOSには、拡張機能呼び出し（INT DCH）に、MS-DOSのドライブ名A～ZをDA/UAに変換するための非公開ファンクションが用意されている。
- IO.SYSのワークエリア0060:006CHからの16バイトにはA～Pに対応するDA/UAが書き込まれていることが知られている。MS-DOS 3.1最終版より前のバージョンのMS-DOSの場合はここを参照する。
- デバイスドライバによっては、前記の方法でそのドライブのDA/UAを取得すると、なんらかの値が得られることがある。しかし、通常デバイスドライバで組み込まれたドライブにはDISK BIOSでアクセスすることはできないので、DA/UAを取得したらDISK BIOSで使用可能な値かどうかをチェックして、DISK BIOSで利用できない値の場合は除外する。

▶プログラミングテクニック

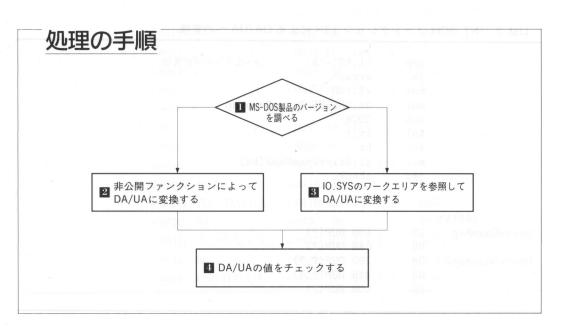
1 MS-DOSの製品バージョンを調べる

MS-DOS 3.1最終版以降のMS-DOSには、拡張機能呼び出し（INT DCH）にドライブ名をDA/UAに変換する非公開ファンクションが用意されている。それ以前のMS-DOSの場合は、IO.SYSのワークエリアにあるDA/UAを参照する。MS-DOS 3.1最終版以降のMS-DOSでもIO.SYSのワークエリアを参照することはできるが、MS-DOS 3.3以降ではSCSIハードディスクや光ディスクをサポートしているため、ドライブ名がA～Pに収まらない場合がある。そこで、拡張機能呼

び出しの中にDA/UA変換機能を持ったMS-DOSのときにはこちらを使う。

まず最初に、MS-DOSのバージョンを調べ、この非公開ファンクションがサポートされているかどうかを調べる。MS-DOSのバージョンを調べるファンクションは、INT DCHのファンクション12Hである。CLに12HをセットしてINT DCHを実行すると、AXに製品バージョン、DXに機種情報が返される（Table 1参照）。

Table 1　MS-DOSの製品バージョンを取得する拡張機能呼び出し

ファンクション	内　容		
INT DCH 12H	MS–DOSの製品番号の取得		
	入力　CL←12H		
	出力　AX←製品バージョン		
	DX←機種情報		
	AX	バージョン	製品名
	不変	2.11、3.1（最終版以外）	下記以外（PC–98LT/HAのROM DOSも含む）
	0001H	3.1最終版	PS98–011
	0002H	3.3、3.3A	PS98–013、015
	0002H	3.3B	PS98–017
	0003H	3.3C	PS98–019
	0004H	3.3D	PS98–1002
	0101H	5.0	PS98–1003

MS-DOS Ver.3.1の最終版以降のバージョンではAXに1以上の値が入る。それより前のバージョンではAXの値は不変である。そこで、あらかじめAXに0000Hをセットしてファンクションを実行し、実行後もAX＝0000HならDA/UAへの変換ファンクションをサポートしていないMS-DOSであると判断する。DA/UAへの変換ファンクションをサポートしているときは、このファンクションを利用し、それ以外はIO.SYSのワークエリアを見る。List 1にこの判別を行うプログラムを示す。

List 1　MS-DOS製品バージョンチェック

```
mov     cl,12H
mov     ax,0000H
int     0DCH
cmp     ax,0000H
je      old_type            ; 新方式が使えない
```

2 非公開ファンクションによってDA/UAに変換する

DA/UA変換を行う非公開のファンクションは、INT DCHのファンクション番号13Hである。CLレジスタに13HをセットしてINT DCHを実行すると、DS:DXの指す領域にMS-DOSドライブ名とDA/UAの対応リストが得られる(List 2参照)。リストの構造はTable 2に示したとおりである。A～ZドライブのDA/UAを取得するため、テーブルの1AH～4DHの内容を用いる。変換が終わったら、DA/UA値がDISK BIOSで利用できる値かチェックする。

List 2　INT DCHファンクション13HによるDA/UAへの変換

```
                cmp     bl,'Z'-'A'          ;A～Zドライブが有効
                ja      error
                mov     cl,13H
                mov     dx,OFFSET DriveDauaMap
                int     0DCH
                shl     bx,1
                inc     bx
                mov     al,DriveDauaMap2[bx]
                jmp     check


        .DATA?
DriveDauaMap    DB      10H DUP(?)
                DB      0AH DUP(?)
DriveDauaMap2   DB      26D DUP(?,?)
                DB      01H DUP(?,?)
                DB      10H DUP(?)
```

Table 2　MS-DOS ドライブ名と DA/UA の対照リストの取得（非公開機能）

ファンクション	内　容
INT DCH　13H	ドライブと DA/UA の値の対応表を求める INT DCH のファンクション 12H をサポートしている MS–DOS で利用できる。 **入力**　CL ← 13H DS:DX ←リスト出力バッファ（60H バイト）の先頭アドレス。 **出力**　DS:DX にリストが書き込まれる。 **リスト出力バッファの内容**

+0000：A:の DA/UA	+0020：00	+0040：00
+0001：B:の DA/UA	+0021：D:の DA/UA	+0041：T:の DA/UA
+0002：C:の DA/UA	+0022：00	+0042：00
+0003：D:の DA/UA	+0023：E:の DA/UA	+0043：U:の DA/UA
+0004：E:の DA/UA	+0024：00	+0044：00
+0005：F:の DA/UA	+0025：F:の DA/UA	+0045：V:の DA/UA
+0006：G:の DA/UA	+0026：00	+0046：00
+0007：H:の DA/UA	+0027：G:の DA/UA	+0047：W:の DA/UA
+0008：I:の DA/UA	+0028：00	+0048：00
+0009：J:の DA/UA	+0029：H:の DA/UA	+0049：X:の DA/UA
+000A：K:の DA/UA	+002A：00	+004A：00
+000B：L:の DA/UA	+002B：I:の DA/UA	+004B：Y:の DA/UA
+000C：M:の DA/UA	+002C：00	+004C：00
+000D：N:の DA/UA	+002D：J:の DA/UA	+004D：Z:の DA/UA
+000E：O:の DA/UA	+002E：00	+004E：00
+000F：P:の DA/UA	+002F：K:の DA/UA	+004F：00
+0010：?	+0030：00	+0050：?
+0011：?	+0031：L:の DA/UA	+0051：?
+0012：?	+0032：00	+0052：?
+0013：?	+0033：M:の DA/UA	+0053：?
+0014：?	+0034：00	+0054：?
+0015：?	+0035：N:の DA/UA	+0055：?
+0016：?	+0036：00	+0056：?
+0017：?	+0037：O:の DA/UA	+0057：?
+0018：?	+0038：00	+0058：?
+0019：?	+0039：P:の DA/UA	+0059：?
+001A：00	+003A：00	+005A：?
+001B：A:の DA/UA	+003B：Q:の DA/UA	+005B：?
+001C：00	+003C：00	+005C：?
+001D：B:の DA/UA	+003D：R:の DA/UA	+005D：?
+001E：00	+003E：00	+005E：?
+001F：C:の DA/UA	+003F：S:の DA/UA	+005F：?

3 IO.SYSのワークエリアを参照してDA/UAに変換する

IO.SYS のワークエリア 0060:006CH からの 16 バイトには A〜P ドライブに対応する DA/UA が書き込まれている。MS-DOS 3.1 最終版より前のバージョンではここを参照して DA/UA を取得する。

A〜P 以外のドライブ名が指定されたときにはエラーを返す。変換が終わったら DA/UA 値が DISK BIOS で利用できる値かチェックする。

List 3 に IO.SYS のワークエリアを参照するプログラムを示す。

List 3　IO.SYS のワークエリアの参照による DA/UA への変換

```
old_type:       cmp     bl,'P'-'A'          ;A〜P ドライブが有効
                ja      error
                mov     ax,0060H            ;MS-DOS のワークエリア
                mov     es,ax
                mov     al,es:[006CH+bx]
```

4 DA/UAの値をチェックする

デバイスドライバで組み込まれたドライブを対象から外すために、DA/UA が DISK BIOS で使用可能な値かどうかをチェックする。デバイスドライバによっては、2、3 の方法で特別な DA/UA（MS-DOS 付属 RAMDISK.SYS では E0H）を返すものがあるためである。Table 3 に示す以外の値が返ってきたら、エラーとする。

Table 3　作成したライブラリが返す DA/UA

値	種　類
70H〜73H	640K バイトフロッピーディスク 両用フロッピーディスクインターフェイスの 640K バイトインターフェイスモード
80H〜83H	SASI ハードディスク
90H〜93H	1M バイトフロッピーディスク 両用フロッピーディスクインターフェイスの 1M バイトインターフェイスモード
A0H〜A7H	SCSI ハードディスク、OD

List 4 に範囲チェックのプログラムを示す。

List 4　DA/UA の範囲チェック

```
check:      cmp     al,70H          ;00H〜6FH → error
            jb      error
            cmp     al,0A8H         ;A8H〜FFH → error
            jae     error
            cmp     al,0A0H         ;A0H〜A7H → ok
            jae     good
            test    al,0CH          ;x4H〜xFH → error
            jnz     error
good:       mov     ah,00H
            ret
error:      mov     ax,00H
```

▶ワンポイント

■ディスクインターフェイスのDA/UA

DA/UA は DISK BIOS でドライブユニットを指定するために用いる値で、MS-DOS でいうところのドライブ名に当たるものである。ただし、DA/UA は物理的なドライブを特定するための値なので、MS-DOS のドライブ名のようにソフトウェアの設定で同じドライブの値が変化してしまうようなことはない。DA/UA は、上位 4 ビット（Device Address）がディスクインターフェイスの種類を、下位 4 ビット（Unit Address）がそのインターフェイスのなかのドライブ番号を表している。Table 4 に DA/UA の値とデバイスの関係を示した。

ハードディスクと両用フロッピーディスクインターフェイスでは、1 つのドライブが 2 種類の DA/UA を持っている。たとえば、80H と 00H は同一の SASI 型ハードディスクの 1 台目であり、93H と 13H は同一の 1M バイトインターフェイスモードの両用フロッピーディスクドライブの 4 台目であるという具合だ。これは、あとで説明するハードディスクのセクタ指定方式や、両用ドライブのメディアタイプの指定のためである。

Table 4　DA/UA の値とディスクドライブの対応

UA/UA	ディスクドライブ
00H～03H	SASI ハードディスク（リニアセクタ番号指定）
10H～13H	両用フロッピーディスク 1M バイトインターフェイスモードの 640K バイトフロッピーディスクアクセスモード
20H～27H	SCSI ハードディスク（リニアセクタ番号指定）
3*x*H	未使用
4*x*H	未使用
50H～53H	320K バイトフロッピーディスク
6*x*H	未使用
70H～73H	640K バイトフロッピーディスク、両用フロッピーディスク 640K バイトインターフェイスモードの 640K バイトアクセスモード
80H～83H	SASI ハードディスク（絶対セクタ番号指定）
90H～93H	1M バイトフロッピーディスク、両用フロッピーディスク 1M バイトインターフェイスモードの 1M バイトアクセスモード
A0H～A7H	SCSI ハードディスク（絶対セクタ番号指定）
B*x*H	未使用
C0H～C7H	SCSI デバイス
D*x*H	未使用
E*x*H	未使用
F0H～F3H	両用フロッピーディスク 640K バイトインターフェイスモードの 1M バイトフロッピーディスクアクセスモード

* DA/UAの下位4ビットは、*x*0＝ドライブの1台目、*x*1＝2台目、…を表す。
* SCSIハードディスクとSCSIデバイスでは下位4ビットはSCSI IDを表す。
* SASIハードディスクの3～4台目はPC-H98のハードディスク内蔵型のみ。
* 両用インターフェイスに接続されるフロッピーディスクドライブのうち、実際に両用として使えるのは内蔵の2台まで。1～2台目と3～4台目のどちらを内蔵として使うかはディップスイッチ1-4で変更できる。
* 両用インターフェイスの外付け1Mバイトフロッピーディスクコネクタにフロッピーディスクドライブを増設したとき、1Mバイトインターフェイスモードのときだけ外付けフロッピーディスクドライブが使用できる。

■両用ドライブとインターフェイスモード

VM 以降の機種に搭載されている両用フロッピーディスクインターフェース(以下両用インターフェイス)は、従来の 1M バイトまたは 640K バイトフロッピーディスクインターフェイスとの互換性を保ちながら 2 種類のメディアを扱えるように拡張されている。

両用インターフェイスは 1M バイトインターフェイスモードと 640K バイトインターフェイスモードの 2 種類のモードを持っていて、インターフェイスモードを変更すると、DA/UA や I/O ポートアドレス、DMA チャネル、外部割り込み番号がそれぞれの専用インターフェイスと同じ状態になる。一般的に、インターフェイスモードは起動時に決定した状態から変更することはない。

両用インターフェイスはどちらのインターフェイスモードでも、アクセスモードを切り替えて 1M バイトフォーマットと 640K バイトフォーマットのディスクにアクセスできるように拡張されている。前に触れたとおりアクセスモードは DA/UA で指定する。インターフェイスモードとア

クセスモードの組み合わせとDA/UAの対応表をTable 5に示す。この表は、1Mバイトインターフェイスモードで1Mバイトフォーマットのディスクにアクセスするときには DA として 9*x*H を、640Kバイトフォーマットのディスクにアクセスするときには 1*x*H を用いることを表している。

Table 5　内蔵両用フロッピーディスクインターフェイスのインターフェイスモード・アクセスモード別対応DA

	1Mバイトフォーマット アクセスモード	640Kバイトフォーマット アクセスモード
1Mバイトインターフェイスモード	9***x***H	1***x***H
640Kバイトインターフェイスモード	F***x***H	7***x***H

Table 6　内蔵両用フロッピーディスクインターフェイスのモード決定方法

条　件	インターフェイスモード
DIP SW 3–1 = ON（固定モード）のとき	
DIP SW 3–2 = OFF（1Mバイト指定）	1Mバイトインターフェイスモード
DIP SW 3–2 = ON（640Kバイト指定）	640Kバイトインターフェイスモード
DIP SW 3–1 = OFF（自動切り換えモード）のとき	
フロッピーディスク以外から起動したとき	
DIP SW 3–2 = OFF（1Mバイト指定）	1Mバイトインターフェイスモード
DIP SW 3–2 = ON（640Kバイト指定）	640Kバイトインターフェイスモード
フロッピーディスクから起動したとき	
起動ディスクが1Mバイトフォーマット	1Mバイトインターフェイスモード
起動ディスクが640Kバイトフォーマット	640Kバイトインターフェイスモード

両用インターフェイスのインターフェイスモードとアクセスモードは、I/OポートのBEHを読むと現在の状態を取得することができる。また、同じポートに書き込むことで変更することもできる。ただし、ディップスイッチ3-1がONのときはポートからのインターフェイスモードの変更は無視され、ハードウェアレベルでディップスイッチ3-2の状態に従う。BEHポートからはディップスイッチ3の1、3の2の状態を取得することもできる（Table 7参照）。

アクセスモードの変更はプログラムから必要に応じて行うことができるが、インターフェイスモードは容易に変更できない。I/OポートBEHやディップスイッチの操作でインターフェイスモードを変更するときは、遷移前のインターフェイスモードで使用していた割り込みを割り込み禁止にしておく必要があり、そうしなければ暴走してしまう。ハードウェア的な切り替えはこれだけで済むのだが、MS-DOS起動後にインターフェイスモードを変更してもMS-DOSはそのことを認識しないし、BIOSレベルでもワークエリアの再設定などが必要になる。したがって、システム起動後にインターフェイスモードを変更するメリットはほとんどないといえるだろう。

Table 7　両用インターフェイスの状態

I/Oポート	Read/Write	意　味		
BEH	Read	両用インターフェイスポート リードモードステータス		
		ビット	意　味	
		bit7〜4	*x*	
		bit3	DSW（DIP SW 3-2の状態）	
			値	意　味
			1	OFF（1Mバイト指定）
			0	ON（640Kバイト指定）
		bit2	FIX（DIP SW 3-1の状態）	
			値	意　味
			1	ON（固定モード…PORT EXC無効）
			0	OFF（自動切換モード…PORT EXC有効）
		bit1	FDD EXC（FDD Mode Exchange）	
			値	意　味
			1	1Mバイトフォーマットアクセスモード
			0	640Kバイトフォーマットアクセスモード
		bit0	PORT EXC（Port Exchange）	
			値	意　味
			1	1Mバイトインターフェイスモード
			0	640Kバイトインターフェイスモード
	Write	両用インターフェイスポート ライトモードチェンジ		
		ビット	意　味	
		bit7〜3	0	
		bit2	EMTON（ポート0094H bit3 MTON有効／無効） （このビットはPC-9801UX以降のみ有効）	
			値	意　味
			1	1Mバイトインターフェイスモード時、常時モータON
			0	1Mバイトインターフェイスモード時、MTONビット有効（モータ制御あり）
		bit1	FDD EXC（FDD Mode Exchange）	
			値	意　味
			1	1Mバイトフォーマットアクセスモード
			0	640Kバイトフォーマットアクセスモード
		bit0	PORT EXC（Port Exchange）	
			値	意　味
			1	1Mバイトインターフェイスモード
			0	640Kバイトインターフェイスモード

ライブラリ

DriveToDaua

解説 MS-DOSのドライブ名をDA/UA（デバイス種別／ユニット番号）に変換する。書式はつぎのとおり。

int **DriveToDaua**(unsigned char *drive*)

drive ドライブ名（'A'～'Z'）

JOIN、SUBST、ASSIGNの設定にかかわらず、起動時のドライブ割り当てに従う

戻り値 ドライブ名と対応するDA/UAの値を返す。

値	意味
0	存在しないドライブ
70～73	640Kバイトフロッピーディスク、両用フロッピーディスクインターフェイスの640Kバイトインターフェイスモード
80～83	SASIハードディスク
90～93	1Mバイトフロッピーディスク、両用フロッピーディスクインターフェイスの1Mバイトインターフェイスモード
A0～A6	SCSIハードディスク、光磁気ディスク

両用フロッピーディスクインターフェイスのドライブでは、現在のアクセスモードにかかわらずインターフェイスモードの本来のDA/UA(1Mバイト=9xH、640Kバイト=7xH）を返す。

SASIやSCSIのハードディスクでは、絶対セクタ番号によるアクセス方式のDA/UA（SASI=8xH、SCSI=AxH）を返す。

拡張フォーマットで領域分割されたハードディスクや光ディスクでは、1台のディスク装置に複数のMS-DOSのドライブが割り当てられる。

サンプル DISKSTAT.C

メモリスイッチの読み出し

メモリスイッチは、周辺装置のモード、拡張装置の有無、BASICのモード等を設定しておくための不揮発性メモリで、PC-9800シリーズの全機種が持っている。しかし、ハイレゾモードやPC-98LT/HAには、BIOSにメモリスイッチへアクセスするファンクションが用意されているが、ノーマルモードには用意されておらず直接メモリを参照するしかない。ここでは、各機種に合わせてメモリスイッチの設定値を読み出す方法を解説する。

▶ポイント

- メモリスイッチからの読み出し方法は、ノーマルモード、ハイレゾモード、PC-98LT/HAで異なるため、機種判別して機種に応じた方法で読み出す。
- ハイレゾモードとPC-98LT/HAではBIOSにメモリスイッチにアクセスするファンクションが用意されているので、それを利用してメモリスイッチの値を読み出す。
- ノーマルモードでは、BIOSにメモリスイッチにアクセスする機能がないので、メモリから直接値を読み出す。

▶プログラミングテクニック

1 機種の判別をする

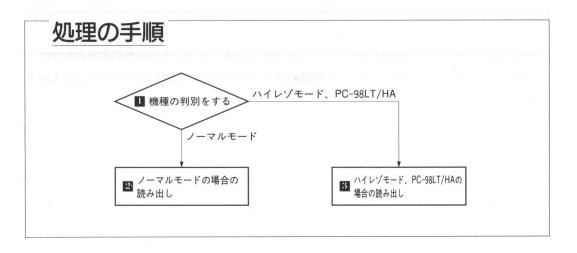
ノーマルモードとハイレゾモードのメモリスイッチの実体はバッテリバックアップされた8バイトの不揮発性メモリで、低位のアドレスから順にSW1〜8の名前がつけられている。ただし、いまのところSW7は利用されていない。SW8は、時計用LSIに年の管理機能を持たないμPD1990を使用している機種で、年を保持するために使用している。ノーマルモード、ハイレゾモード両用機では、モードごとに独立したメモリスイッチが用意されている。

ノーマルモードとハイレゾモードのメモリスイッチはテキストVRAMのアトリビュートの最後部付近に割り当てられている。通常は書き込みが禁止されていて、不用意に書き換えられないようになっている。この書き込み禁止は、I/Oポートにアクセスすれば解除できる。ただし、ハイレゾモードにはメモリスイッチの書き換えと読み出しのためのBIOSが用意されているのでこれを

利用することもできる。ノーマルモードにはこのBIOSがないので、プログラムで直接メモリをアクセスして書き換えや読み出しを行わなければならない。

PC-98LT/HAのメモリスイッチは内蔵辞書の学習情報などを記憶するための学習用RAMエリアのバンク3に存在する。こちらは書き込みが禁止されていないかわりに、サムチェックを行っている。チェックサムが合わないとリセットしたときにメモリ内容が初期化されてしまう。ハイレゾモードと同様にメモリスイッチの書き換えと読み出しのためのBIOSが用意されているので、アクセスするときにはこれを利用するようにする。学習用RAMエリアはバンク切り替え可能になっているので、メモリスイッチを参照するだけでもBIOSを使うべきである。なお、PC-98LT/HAではSW1〜8以外にもメモリスイッチがあり、HANDY MENUなどで使用されている。ノーマルモード、ハイレゾモード、PC-98LT/HAの、メモリスイッチの位置を、Table 1に示す。

Table 1　メモリスイッチのメモリ上の位置

	ノーマルモード	ハイレゾモード	PC-98LT/HA
スイッチ0	なし	なし	D000:3800H
スイッチ1	A000:3FE2H	E000:3FE2H	D000:3801H
スイッチ2	A000:3FE6H	E000:3FE6H	D000:3802H
スイッチ3	A000:3FEAH	E000:3FEAH	D000:3803H
スイッチ4	A000:3FEEH	E000:3FEEH	D000:3804H
スイッチ5	A000:3FF2H	E000:3FF2H	D000:3805H
スイッチ6	A000:3FF6H	E000:3FF6H	D000:3806H
スイッチ7	A000:3FFAH	E000:3FFAH	D000:3807H
スイッチ8	A000:3FFEH	E000:3FFEH	D000:3808H
スイッチ9	なし	なし	D000:3809H
：			：
スイッチ255			D000:38FFH

このように、ノーマルモード、ハイレゾモード、PC-98LT/HAでメモリスイッチからの読み出し方法が異なるので、最初に機種の判別を行う。ハイレゾモードかPC-98LT/HAならばBIOSがメモリスイッチにアクセスする機能を持っているのでそれを利用する。ノーマルモードならばメモリから直接値を読み込む処理をする。

判別のために必要な情報は、**システム共通域0000:0500Hのbit12、bit11、bit0**から得られる(Table 2参照)。

Table 2　機種判別のために参照するシステム共通域

アドレス	意　味	
0000:0500H	BIOS制御用フラグ0 (BIOS_FLAG)	
	bit0	機種情報1 (PC-9801初代/E/F/M、PC-98LT/HA判別)
	1	その他
	0	PC-9801初代/E/F/M、PC-98LT/HA/ハイレゾモード
	bit12	機種情報3 (PC-9801U2、PC-98LT/HA判別)
	1	PC-9801U2、PC-98LT/HA
	0	その他
	bit11	ハイレゾモード／ノーマルモード判別
	1	ハイレゾモード
	0	ノーマルモード

ここで得られた、機種情報1、3、4からTable 3のように機種を判別する。

Table 3　機種情報と機種の対応

機　種	機種情報1 (bit0)	機種情報3 (bit12)	機種情報4 (bit11)
PC-98LT/HA	0	1	0
ノーマルモード	0/1	0/1	0
ハイレゾモード	0	0	1

機種情報1、3が0と1の組み合わせならばPC-98LT/HAである。それ以外の組み合わせならノーマルモードかハイレゾモードなので、機種情報4でどちらかを判定する。プログラムをList 1に示す。

List 1　機種判別

```
mov     ax,0000H            ; ES ← 0000（システム共通域）
mov     es,ax
mov     ax,es:[0500H]
and     ax,1001H            ; LT,HA 判定
cmp     ax,1000H
je      BiosUse             ; LT,HA ならジャンプ
test    es:[0501H],BYTE PTR 08H ; ハイレゾモード判定
jnz     BiosUse             ; ハイレゾモードならジャンプ
```

2 ノーマルモードの場合の読み出し

ノーマルモードのメモリスイッチはA000:3FE2Hから始まり、4バイト間隔で8バイト存在する。与えられたスイッチ番号から1を引き、4倍した値に3FE2Hを加えるとメモリアドレスが算出できるので、そこから読み出した値を返す。プログラムをList 2に示す。

List 2　ノーマルモードの場合のメモリスイッチの読み出し

```
; ** ノーマルモードの処理 **
mov     ax,0A000H           ; テキスト VRAM セグメント
mov     es,ax               ; ES ←メモリスイッチのセグメント
mov     bh,00H              ; 3FE2H に (number-1)AND 7 を
mov     bl,number           ; 4 倍した値を加算すると目的のメモリ
dec     bl                  ; スイッチのアドレスになる
and     bl,07H
add     bx,bx
add     bx,bx
mov     al,es:[3FE2+bx]     ; メモリスイッチ読み出し
mov     ah,0
ret
```

3 ハイレゾモード、PC-98LT/HA の場合の読み出し

前述のとおり、ハイレゾモードとPC-98LT/HAではBIOSでメモリスイッチにアクセスできるので、それを利用してメモリスイッチの値を読み出す（Table 4 参照）。

Table 4　メモリスイッチアクセスのための BIOS（ハイレゾモード、PC-98LT/HA のみ）

ファンクション	内　容
INT 18H 21H	メモリスイッチの読み出し 入力　AL＝スイッチ番号（1～8） 出力　DL＝スイッチの内容

プログラムを List 3 に示す。

List 3　ハイレゾモードや PC-98LT/HA でのメモリスイッチの読み出し

```
BiosUse:        mov     ah,21H          ; メモリスイッチ読み出し
                mov     al,number
                int     18H
                mov     al,dl
                mov     ah,0
                ret
```

ライブラリ

GetMemorySwitch

解説　指定したメモリスイッチの値を読む。書式はつぎのとおり。

int **GetMemorySwitch**(int *number*)

number　メモリスイッチの番号（1～8）

戻り値　メモリスイッチの内容

サンプル　MSW.C

COLUMN

非公開機能の解析方法～システム共通域

システム共通域は、一般にはその一部分しか公開されていない。巻末の資料集には筆者が独自に調査した非公開情報を盛り込んであるが、この調査の手法について簡単に紹介する。

システム共通域は大きく分けて、機種情報（機能の有無、使用デバイスの種類など）の格納、BIOSの動作モードの格納、BIOSの作業域の３種類の用途に使用されている。このなかの非公開部分を解析するには次のような手法をとる。

まず、システム共通域をファイルに保存するプログラムを作成して、デバイスドライバ等を一切組み込まないMS-DOSのAUTOEXEC.BATでこのプログラムを実行する。ディップスイッチなどを切り替えてこれを繰り返し、システム共通域のどこが変化するかを調べれば、変化した部分の用途がわかることになる。拡張ボードの有無でも変化する場合がある。また、いろいろな機種でシステム共通域を取得した結果を比較すれば、機種ごとに異なるハードウェアの有無、種類などを示すフラグを見つける手掛かりを得ることができる。

さらに、このようにして推定した内容が本当に正しいかどうかを確認するため、BIOS ROMやITF（Initial Test Firmware） ROMを逆アセンブルする。ITFとはシステム起動時にハードウェアの動作テストや初期化を行うためのROMである。ハードウェアの有無、種類などを示すフラグの多くはITFでセットされる。BIOS、ITFのほか、その情報を利用しそうなMS-DOSのデバイスドライバなどを逆アセンブルしても有用な情報が得られることがある。

逆アセンブルするときは、まず調査したいアドレスを示すコード列を検索して、見つかったアドレスの周辺を逆アセンブルすると効率的である。たとえば、0000:0480Hにアクセスしている部分を調べるには、80H、04Hの並びのコードを検索すればよい。ただし、システム共通域のうちテーブルとして利用している部分では、配列のようにメモリアクセスを行うため、通常ベースアドレスの値しかコード中に現れないので、このような手法は使えないことになる。目的とするメモリアクセスを行っている部分が見つかったら、そのルーチンが推測した動作と矛盾がないかどうかを再度検討する。

また、関連しそうな動作をするBIOSファンクションのエントリから逆アセンブルを進め、BIOSファンクションの動作などから意味を推定するようなことも行う。

メモリスイッチへの書き込み

メモリスイッチに値を書き込む方法を解説する。この場合も、ハイレゾモードやPC-98LT/HAには、メモリスイッチを設定するBIOSファンクションが用意されているが、ノーマルモードには用意されていないので、直接メモリに書き込まなければいけない。しかし、通常はメモリスイッチは書き込みが禁止されているので、これを解除する方法もあわせて解説する。

▶ポイント

- ●メモリスイッチへの書き込み方法はノーマルモード、ハイレゾモード、PC-98LT/HAで異なるため、機種判別して機種に応じた方法で書き込む。
- ●ハイレゾモードとPC-98LT/HAはBIOSがメモリスイッチにアクセスする機能を持っているので、それを利用してメモリスイッチに値を書き込む。
- ●ノーマルモードではBIOSがメモリスイッチにアクセスする機能を持っていないので、メモリに直接、値を書き込む。通常メモリスイッチは書き込みが禁止されているので、書き込みの前にこれを解除し、書き込んだ後再び書き込み禁止にしておく。

▶プログラミングテクニック

1 機種の判別をする

ノーマルモード、ハイレゾモード、PC-98LT/HAでメモリスイッチへの書き込み方法が異なるので、読み出しの場合と同様に最初に機種の判別を行う。ハイレゾモードかPC-98LT/HAならばBIOSがメモリスイッチにアクセスする機能を持っているのでそれを利用する。ノーマルモードならばメモリに直接、値を書き込む処理をする。その詳細はすでに「メモリスイッチの読み出し」で述べた。プログラムをList 1に示す。

List 1　機種判定用のプログラム

```
mov     ax,0000H                    ; ES ← 0000 (システム共通域)
mov     es,ax
mov     ax,es:[0500H]
and     ax,1001H                    ; LT,HA 判定
cmp     ax,1000H
je      BiosUse                     ; LT,HA ならジャンプ
test    es:[0501H],BYTE PTR 08H ; ハイレゾモード判定
jnz     BiosUse                     ; ハイレゾモードならジャンプ
```

2 ノーマルモードの場合の書き込み

ノーマルモードのメモリスイッチはA000:3FE2Hから始まり、4バイト間隔で8バイト存在する。与えられたスイッチ番号から1を引き、4倍した値に3FE2Hを加えるとメモリアドレスが算出できるので、そこに値を書き込む。

ただし、通常メモリスイッチは書き込みが禁止されているので、書き込みの前にI/Oポート68Hにアクセスしてこれを解除し、書き込んだ後再び書き込み禁止にしておく必要がある(Table 1参照)。プログラムをList 2に示す。

Table 1　メモリスイッチの書き込み禁止状態の設定（ノーマルモード、ハイレゾモード）

I/Oポート	意　味	
68H	メモリスイッチの書き換えの禁止と許可	
	値	意　味
	0DH	書き換え許可
	0CH	書き換え禁止

List 2　ノーマルモードの書き込み

```
mov     ax,0A000H          ; テキスト VRAM のセグメント
mov     es,ax              ; ES ←メモリスイッチのセグメント
mov     bh,00H             ; 3FE2H に (number-1)AND 7 を 4 倍した値を
mov     bl,number          ; 加算すると目的のメモリスイッチの
dec     bl                 ; アドレスになる
and     bl,7
add     bx,bx
add     bx,bx
; メモリスイッチ書き換え許可
mov     al,0DH
out     68H,al
; メモリスイッチに書き込み
mov     al,value           ; メモリスイッチへ書き込み
mov     es:[bx+3FE2H],al
; メモリスイッチ書き換え禁止
mov     al,0CH
out     68H,al
ret
```

3 ハイレゾモード、PC-98LT/HA の場合の書き込み

読み出しの場合と同様、ハイレゾモードとPC-98LT/HAではBIOSファンクションでメモリスイッチにアクセスできるので、それを利用してメモリスイッチに値を書き込む(Table 2 参照)。プログラムをList 3 に示す。

Table 2　メモリスイッチ BIOS（ハイレゾモード、PC-98LT/HA）

ファンクション	内　容
INT 18H 22H	メモリスイッチの書き込み 入力　AL＝スイッチ番号 (1～8) 　　　DL＝スイッチの内容 出力　なし

List 3　ハイレゾモードとPC-98LT/HA でのメモリスイッチの書き換え

```
BiosUse:        mov     ah,22H          ; メモリスイッチ書き換え
                mov     al,number
                mov     dl,value
                int     18H
                ret
```

▶ワンポイント

■ディップスイッチとメモリスイッチ

システムの各種設定をするスイッチには、ほかにディップスイッチがある。ディップスイッチは本来ハードウェアの電気的な変更を行うために設けられているものだが、一部の設定以外はソフトウェア的な設定のために使用されている。ディップスイッチの操作で直接電気的な変更が行われるのは、DIP SW 1-2 のスーパーインポーズ、1-4 のフロッピーディスクの番号指定、1-5と6のRS-232C モード、3-1 と 2 の両用フロッピーディスクのモード等で（機種によっては 3-6 のRAM KILL も電気的な変更をともなう)、そのほかはソフトウェアがディップスイッチの状態を読み取り、それにしたがってソフトウェアの設定を行っているだけである。

ライブラリ

SetMemorySwitch

解説　指定したメモリスイッチに値を書く。書式は次のとおり。

void **SetMemorySwitch**(int *number*, int *value*)

number　メモリスイッチの番号（1〜8）
value　設定する値

戻り値　なし

サンプル　MSW.C

拡張ROM領域のメモリ存在調査

ソフトウェアのコントロールで、拡張ROM領域にメモリを出現させるような拡張ボードを制御する場合、メモリを出現させようとする領域にメモリが存在していないことを前もって確認しておく必要がある。直接メモリの存在を調べる機能がないので、いくつかの方法を組み合わせて、ROMやRAMの存在可能性を1つずつ潰していくしかない。

ここでは、任意のアドレスにROMやRAMが存在しないことを検出する方法を解説する。

▶ポイント

- ノーマルモードでは、メモリが存在しないアドレスにアクセスして値を読むと、データバスがプルアップされているため全ビットが1になりFFHが得られる。
- PC-9801の拡張ROM領域には4Kバイト単位でメモリが実装される。このためメモリの存在調査も4Kバイトを単位に行う。4KバイトにわたってFFHで埋められているROMを実装するという無意味な状態は通常ありえないので、4KバイトがすべてFFHならそこにROMは存在しないと判断する。
- 初期のハイレゾ機では、メモリが存在しないアドレスから普通に値を読んでもFFHが得られない。そこで、メモリの読み出し方を工夫してFFHを得られるようにする。
- FFHが読み出されたとしても、RAMが存在し、FFHが書き込まれている可能性もある。メモリ非存在と誤認しないように、RAMではないことも確認する必要がある。
- PC-98LT/HAでは、データバスにプルアップ抵抗の代わりにアクティブターミネータが使用されている。このためメモリが存在しないアドレスから値を読んでもFFHが得られない。ROMありと誤認しないようにこのための検査を行う。

▶プログラミングテクニック

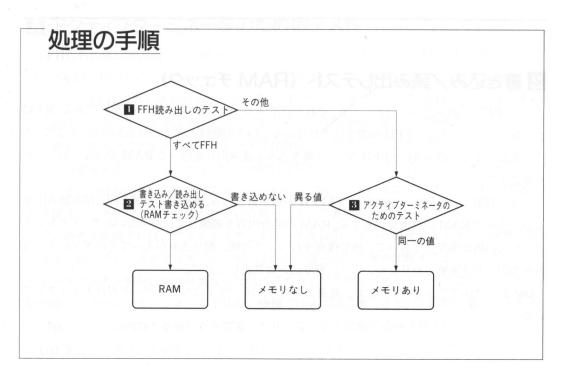
1 FFH読み出しのテスト

ノーマルモードでは、ROMやRAMの存在しないアドレスにアクセスすると、データバスがプルアップされているため全ビットが1である値（FFH）が得られる。なお、プルアップとは、信

号線を抵抗を介してハイレベルの電圧に接続しておくことで、バスの安定した動作のために必要なものである。

ところが、初期のハイレゾ機（PC-98XA/LX）ではメモリが存在しないアドレスに普通にアクセスして値を読んでもFFHが得られない。得られる値がメモリの読み出しを行う命令より数バイト先にある命令コードの値と一致することから、バスの静電容量やプルアップ抵抗の値などの加減で、CPUがプリフェッチしたときのデータバスの状態がそのまま保存されているのではないかと考えられるが、詳細は不明である。そこでいろいろ実験してみた結果、REP LODS命令を使って2回続けざまにメモリを読み出すと、2回目以降の読み出しではFFHが得られることがわかった。REP LODS命令で連続したメモリの読み出しをしたときには命令フェッチサイクルが入らないので、データバスがHIGHレベルに引き上げられるのに十分な時間、バスがハイインピーダンス状態を保つためであろう。

通常、PC-9801の拡張ROM領域では4Kバイト単位でメモリが実装される（アドレスの境界も4Kバイトである）。これは、システム共通域にD0000H～DFFFFH（ハイレゾモードはE6000H～EFFFH）の拡張ROMを管理するためのテーブル（0000：04D0H～）があって、4Kバイトごとに拡張ROMのIDが登録できるような構造になっていることから来ている。ハードウェア的にはもちろん任意のサイズのROMを実装することができるのだが、実際のメモリ搭載ボードはごく一部の例外を除きシステム共通域のテーブルの構造に倣って4Kバイト単位でメモリを搭載している。

このことから、メモリの有無の検査は4Kバイト単位で行えばよいことになる。1単位（4Kバイト）の領域全体を調べて、すべてFFHが読み出されたらROMなしと判断できる。1単位にわ

たってFFHが書き込まれているROMを実装することは可能ではあるが、そのような無意味なROMが実装されることは通常あり得ないので考慮する必要はない。

List 1にFFH読み出しのテストを行うプログラムを示す。なお、このプログラムではREP LODSW命令を使って、FFFFHとの比較を行っている。また、実際には1ワードおきにしか検査していないが、サンプリング数は十分多く実用上問題はない。

List 1　FFHの読み出しテスト

```
AREA_SIZE EQU 1000H         ;4KB                ; 1つのROM領域のサイズ
          ;--- FFH読み出しのテスト
                  ; 全部FFH → 書き込み／読み出しテスト（RAMチェック）
                  ; その他の値が読めた → アクティブターミネータのためのテスト
                  mov     cx,AREA_SIZE / 4
                  mov     si,0000H              ; 検査開始番地のオフセット
                  cld
RomTestLoop:      push    cx
                  mov     cx,0002H
                  cli                           ; 割り込み禁止
                  rep     lodsw                 ; 2ワード連続読み込み
                  sti
                  pop     cx
                  cmp     ax,0FFFFH             ; 読み出した値がFFFFHでなかったら
                  jne     AtTest                ; その他の値が読めた → ATのための確認へ
                  loop    RomTestLoop
                  ;                             ; 全部FFH → RAMかどうかの確認へ
```

2 書き込み／読み出しテスト（RAMチェック）

さて、FFHが読み出されたとしても、それだけでメモリなしと判断するのは早計である。RAMが存在していて、そこにFFHが書き込まれているという可能性も考えられるからだ。そこで、さらに異なる2つの値(00HとFFHなど)の書き込みと読み出しを行ってRAMではないかどうかを確認する。

テスト対象のアドレスに0000HとFFFFHの値を書き込み、書き込んだ値と同じ値が読み出せるかどうかでRAMの有無を判断する。RAMの場合内容を破壊してはならないので、テストするアドレスの値は保存しておいて、後で復帰を行う。この間、割り込みが入ってそのアドレスの内容を参照したり変更したりしないように割り込みは禁止しておく。

List 2にプログラムを示す。なお、高速化のため、16バイトにつき1ワードだけチェックしている。

List 2　RAMの有無のテスト

```
AREA_SIZE EQU 1000H      ;4KB       ;1つのROM領域のサイズ
INTERVAL  EQU 10H        ;16bytes   ;調査間隔（高速化のためすべては調べない）
        ;--- 書き込み／読み出しテスト（RAMチェック）
                ;書き込んだ値が読み出せた（RAMあり）→メモリあり
                ;書き込んだ値が読み出せない→メモリなし
                mov     cx,AREA_SIZE / INTERVAL
                mov     si,0000H            ;SI ← 0000H
                mov     ax,si               ;AX ← 0000H
                mov     bx,ax
                dec     bx                  ;BX ← FFFFH
RamTestLoop:    cli                         ;割り込み禁止
                push    [si]                ;[SI]の値を保存
                mov     [si],ax             ;[SI]に0000Hを書き込む
                cmp     [si],ax             ;[SI]の値を0000Hと比較
                jne     RamTest1
                mov     [si],bx             ;[SI]にFFFFHを書き込む
                cmp     [si],bx             ;[SI]の値をFFFFHと比較
RamTest1:       pop     [si]                ;[SI]の値を復帰
                sti                         ;割り込み許可
                je      MemoryPresent       ;一致したらRAMあり
                add     si,INTERVAL
                loop    RamTestLoop
                jmp     MemoryAbsent        ;RAMなし
```

3 アクティブターミネータのためのテスト

FFH以外の値が読み出された場合にもさらにチェックが必要である。PC-98LT/HAではプルアップ抵抗の代わりにアクティブターミネータと呼ばれる素子を使用しているので、メモリが存在しなくてもFFH以外の値が読めることが多い。アクティブターミネータは接続された信号線の状態に追従してダイナミックにプルアッププルダウンの作用を切り替えて消費電力を小さくするための素子である。アクティブターミネータでは直前のデータバスの状態が残るので、メモリが存在しない領域を読み出したときは、多くの場合CPUがプリフェッチしたコードの値が保持されていて、それが読み出されることになる。したがって、同じアドレスにアクセスしても、読み出される値はプログラムコードの置かれている部分によって異なる。

この特性を利用すれば、アクティブターミネータを採用している機種でもメモリの有無を判別できる。同じアドレスをプログラムの別の2か所で読み出した値を比較すると、アクティブターミネータでメモリがない場合は異なった値が読める。アクティブターミネータでもメモリがある場合や、アクティブターミネータではない機種では必ず同じ値となる。この間、割り込みが入って当該アドレスの内容を参照したり変更したりしないように割り込みは禁止しておく。

List 3にプログラムを示す。今回も、16バイトにつき1ワードだけチェックしている。

List 3　アクティブターミーネータのテスト

```
            ;--- アクティブターミネータ (AT) のためのテスト
AtTest:          mov     cx,AREA_SIZE / INTERVAL
                 mov     si,0000H
AtTestLoop:      cli                        ; 割り込み禁止
                 mov     ax,[si]            ; [SI] を AX に読み込む
                 cmp     ax,[si]            ; 再び [SI] を読み AX と比較
                 sti                        ; 割り込み許可
                 je      MemoryPresent      ; 一致したらメモリあり
                 add     si,INTERVAL
                 loop    AtTestLoop
;                jmp     MemoryAbsent       ; 一致しなかったらメモリなし

MemoryAbsent:    mov     ax,0000H           ; メモリなし
                 ret
MemoryPresent:   mov     ax,0001H           ; メモリあり
                 ret
```

ワンポイント

■メモリマップドI/Oについて

まれではあるがメモリマップド I/O の拡張ボードがある。メモリマップド I/O があった場合、RAM の存在調査の中で FFFFH や 0000H を I/O ポートに書き込むことになるので、このプログラムを実行した場合、ボードが異常動作する可能性がある。

ライブラリ

ExistExtRom

解説　拡張メモリ領域のメモリ存在調査を行う。書式はつぎのとおり。

int **ExistExtRom**(unsigned int *CheckSegment*)

CheckSegment　調査するメモリのセグメント

CheckSegment で指定されたセグメントの先頭 4K バイトの範囲で、メモリが存在するかどうかを調べる。

戻り値　戻り値と意味の関係はつぎのとおり。

戻り値	意味
0	メモリなし
1	メモリあり

サンプル　CHECKS.C

COLUMN

非公開機能の解析方法〜I/Oポート

I/Oポートは、そのすべてが『テクニカルデータブック』等で公開されているわけではない。VMまでの機種のI/Oポートはおそらくすべて公開されていたが、PC-9801VX/RA、PC-H98と世代を経るにしたがって非公開なI/Oポートが増えてきている。巻末の資料集には筆者が独自に調査した非公開I/Oポートの情報も盛り込んであるが、この調査の手法について簡単に紹介する。

非公開のI/Oポートの解析は、BIOSやITF、ハードウェアにアクセスするようなデバイスドライバを逆アセンブルしてその存在を知ることから始める。I/Oポートはメモリと違って連続したアドレスに配置されているとは限らないので、既存のプログラムが使用していないものはその存在すら見つけられない可能性が高い。ただし、I/Oアドレスの配置にはある程度の規則性があるので、I/Oポートがいくつか見つかったあとは、配置の規則性から存在を推定できることもある。

新規に追加されるI/Oポートは、下位8ビットだけでは指定できないアドレスに配置されることが多いので、ポートアドレスがわかってしまえばシステム共通域の調査方法と同様に、プログラム中のMOV DX, 〈address〉のコード列を検索して、I/Oポートにアクセスしている部分のほとんどを探し出すことは容易になる。プログラム中では、I/Oポートから読み出した値を加工してシステム共通域にセットしたり、逆にシステム共通域の値を加工してI/Oポートに書き込むようなことを行っている場合がある。それが既知のシステム共通域であれば、そのI/Oポートの働きを推定するのは比較的容易だろう。また、BIOSファンクションのエントリから逆アセンブルを進め、BIOSファンクションの動作などから機能を推定することもできる。

プログラムを逆アセンブルした結果、着目したI/Oポートの機能の見当がついたときは、そこに対する書き込みや読み出しを行って、推定したとおりの動作をするかどうかを確認する。

このような作業を行って、未知のI/Oポートの機能を解明するのである。しかしながら、PC-H98やNOTEシリーズで追加されたI/Oポートの中にはかなり複雑な動作をするものがあるので、その機能を完全に解析するのは極めて困難なことだと思われる。

プリンタの状態調査

プリンタには、印字ができるオンライン状態のほかに、バッファがいっぱいでデータを受けつけない BUSY 状態や、オフライン状態などがある。一般のユーザが利用するプログラムでは、こうした状態をユーザの責任で管理させるのではなく、その状態に応じてユーザにメッセージを出すなどの対応をしたほうがよい。ここでは、ソフトウェアから接続されているプリンタの状態を調査する方法を解説する。

▶ポイント

- ハイレゾモードや PC-H98 など、フルセントロニクスのプリンタインターフェイスを搭載した機種では BIOS を使ってプリンタの詳細な状態が取得できる。
- プリンタインターフェイスが簡易セントロニクス専用の機種ではプリンタの詳細な状態を取得できないので、つぎのように $\overline{\text{BUSY}}$ 信号のふるまいからその状態を推定する。
- データ送信可能状態では、$\overline{\text{BUSY}}$ 信号は通常インアクティブである。また、プリンタにデータを送信すると、その直後一瞬だけ $\overline{\text{BUSY}}$ 信号がアクティブになる。
- オフライン状態では、$\overline{\text{BUSY}}$ 信号はアクティブ状態を継続している。
- プリンタ未接続または電源 OFF 状態なら、データを送信しても $\overline{\text{BUSY}}$ 信号はまったくアクティブにならない。
- プリンタの状態を調べるために送信するデータでプリンタが無用な動作をしてはならないので、データとして無動作コードの DC1（11H）を使用する。

▶プログラミングテクニック

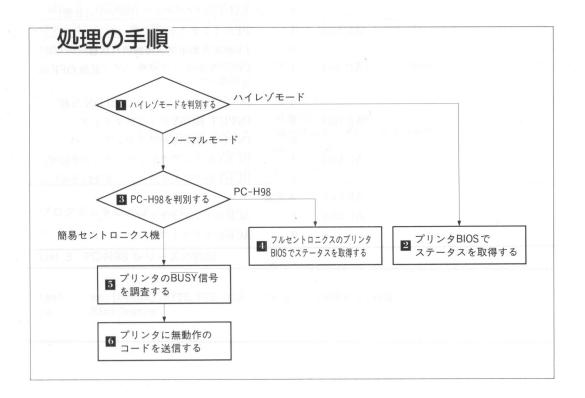
1 ハイレゾモードを判別する

ハイレゾモードのときのプリンタインターフェイスはフルセントロニクスモードなので、BIOS コマンドを使って容易にプリンタの詳細な状態が取得できる。ハイレゾモードかどうかは、**システム共通域 0000:0501H の bit 3** をチェックして判断する（Table 1、List 1 参照）。

Table 1　ハイレゾモード判別のためのシステム共通域

アドレス	意　味		
0000:0501H	BIOS 制御用フラグ（BIOS_FLAG）		
		bit3	ハイレゾモードとノーマルモードの判定
		1	ハイレゾモード（フルセントロ）
		0	ノーマルモード（簡易セントロまたはフルセントロ）

List 1　ハイレゾモードの判別

```
mov     ax,0000H                    ;ES ← 0000 (システム共通域)
mov     es,ax
test    es:[0501H],BYTE PTR 08H ; ハイレゾ／ノーマル判定
jz      Normal
```

2 プリンタ BIOS でステータスを取得する

ハイレゾモードと判断したときは、プリンタ BIOS のステータス取得 (12H) を実行し、AH レジスタの値を戻り値としてリターンする（Table 2 参照)。

Table 2　プリンタのステータス

ファンクション	機　能
INT 1AH AH＝12H	ステータス情報の取得（フルセントロニクスモード時）

入力　なし

出力　AX＝プリンタステータス

レジスタ	値	意　味
AH	00H	データ送信可能（オンライン）状態
	01H	BUSY 状態
	02H	タイムアウトで送信不可能だった
	03H	オフライン状態
	04H	ペーパーエンド
	05H	プリンタ未接続または電源 OFF
AL bit7	1	$\overline{\text{SELECT}}$がインアクティブ（オフライン状態）
	0	$\overline{\text{SELECT}}$がアクティブ（オンライン状態）
AL bit6	1	$\overline{\text{FAULT}}$がインアクティブ
	0	$\overline{\text{FAULT}}$がアクティブ（印字不可状態）
AL bit5	1	$\overline{\text{PE}}$がインアクティブ
	0	$\overline{\text{PE}}$がアクティブ（用紙切れ状態）
AL bit4	1	$\overline{\text{DC+5V}}$がインアクティブ（電源 OFF 状態）
	0	$\overline{\text{DC+5V}}$がアクティブ（電源 ON 状態）
AL bit3	1	INPUT $\overline{\text{BUSY}}$がインアクティブ
	0	INPUT $\overline{\text{BUSY}}$がアクティブ
AL bit2	1	$\overline{\text{BUSY}}$がインアクティブ（データ受信可）
	0	$\overline{\text{BUSY}}$がアクティブ（データ受信不可）
AL bit1	未定義	
AL bit0	1	$\overline{\text{ACK}}$がインアクティブ
	0	$\overline{\text{ACK}}$がアクティブ

プログラムをList 2に示す。

List 2　プリンタBIOSのステータスの取得

```
mov     ah,12H          ; ステータス取得
int     1AH
mov     al,ah
mov     ah,00
ret
```

3 PC-H98を判別する

PC-H98（ノーマルモード専用のPC-H98Sも含む）は、ノーマルモードのときでもフルセントロニクスインターフェイスモードを利用できるように設計されている。プリンタインターフェイスを常時フルセントロニクスモードに設定することもできるが、簡易セントロニクスモードのときでも、新設されたコマンドでフルセントロニクスのステータスを取得することができる。

フルセントロニクスサポートかどうかは**システム共通域0000:0458Hのbit 1**をチェックして判断する（Table 3参照）。

Table 3　PC-H98フルセントロニクスモード判別

アドレス	意　味	
0000:0458H	NESA採用機の情報	
	bit2	**ノーマルモード時のフルセントロニクスと簡易セントロニクスモードとの識別**
	1	ノーマルモード時フルセントロニクスモード
	0	ノーマルモード時簡易セントロニクスモード
	bit1	**ノーマルモード時のフルセントロニクスのサポート**
	1	ノーマルモード時フルセントロニクスモードサポートあり
	0	ノーマルモード時フルセントロニクスモードサポートなし

プログラムをList 3に示す。

List 3　PC-H98シリーズの判別

```
test    es:[0458H],BYTE PTR 02H ; フルセントロ可能ビット検査
jz      MiniCentro
```

4 プリンタBIOSでフルセントロニクスのステータスを取得する

フルセントロニクスサポートと判断したときは、プリンタBIOSのステータス情報の取得ファンクション（18H）を実行し、AHレジスタの値を戻り値としてリターンする（Table 4 参照）。

Table 4　プリンタのセンス

ファンクション	内　容
INT 1AH 18H	ステータス情報の取得（PC–H98のみ） 現在のモード（簡易／フルセントロニクス）にかかわらず、フルセントロニクスモードのステータスが得られる。 入力　なし 出力　AX＝プリンタステータス

プログラムをList 4に示す。

List 4　フルセントロニクスのステータスの取得

```
mov     ah,18H          ; センス
int     1AH
mov     al,ah
mov     ah,00
ret
```

5 プリンタの $\overline{\mathrm{BUSY}}$ 信号を調査する

プリンタインターフェイスが簡易セントロニクス専用の場合は、プリンタの詳細な状態を取得できないので、$\overline{\mathrm{BUSY}}$ 信号のふるまいからその状態を推定する。$\overline{\mathrm{BUSY}}$ 信号の状態は**プリンタインターフェイスの8255Aポート B（I/Oポート42H）のbit 2から取得できる**（Table 5 参照）。

Table 5　プリンタインターフェイスの $\overline{\mathrm{BUSY}}$ 信号取得

アドレス	意　味	
0042H	プリンタインターフェイスの8255A　ポートB	
	bit2	$\overline{\mathrm{BUSY}}$ 信号
	0	プリンタインターフェイスの $\overline{\mathrm{BUSY}}$ 信号アクティブ （プリンタはデータ受信不可）
	1	プリンタインターフェイスの $\overline{\mathrm{BUSY}}$ 信号がインアクティブ （プリンタはデータ受信可）

$\overline{\text{BUSY}}$信号は、プリンタがデータ入力を受け付けられない状態であることをCPU側に知らせるための信号である。この信号はプリンタ内部のデータバッファがいっぱいになるとアクティブになり、プリンタが印字を行ってデータバッファに空きができるとインアクティブになる。また、データバッファに空きがあっても、データを受け取った直後は一瞬アクティブになる。プリンタの電源がOFFであるかプリンタが接続されていないときには常にインアクティブになっている。

$\overline{\text{BUSY}}$信号のこの特性を利用して、プリンタの状態（オンライン状態／オフライン状態／プリンタ未接続または電源OFF）を本体から調査する。データ送信可能状態では$\overline{\text{BUSY}}$信号は通常インアクティブだが、プリンタにデータを送信するとその直後に一瞬だけ$\overline{\text{BUSY}}$信号がアクティブになる。オフライン状態では$\overline{\text{BUSY}}$信号はアクティブ状態が継続している。プリンタ未接続または電源OFF状態なら、データを送信しても$\overline{\text{BUSY}}$信号はまったくアクティブにならない。

そこで、まず$\overline{\text{BUSY}}$信号がアクティブであるかどうかを調べ、アクティブならばオフラインであると判断する。プログラムをList 5に示す。

List 5　プリンタインターフェイスの$\overline{\text{BUSY}}$信号取得

```
in      al,0042H        ;BUSY#信号チェック (0でアクティブ)
test    al,04H          ;
jz      offline
```

6 プリンタに無動作のコードを送信する

$\overline{\text{BUSY}}$信号がインアクティブだったら、プリンタは電源ONでオンラインなのか、電源OFF（または未接続）なのかを判断する。その方法は、プリンタに無動作のコードを送り、一定時間の間に$\overline{\text{BUSY}}$信号が一瞬でもアクティブになるかどうかで行う。プリンタが接続されていないかプリンタの電源がOFFのときは、コードを送っても$\overline{\text{BUSY}}$信号はアクティブにならない。

無動作のコードとしては、DC1（11H）を用いる。DC1はPC-PR系のシリアルプリンタではオンラインへの切り替えコマンドとして割り当てられている。PC-PR系のページプリンタやESC/P系では無動作である。

ここではDC1を送信するのにBIOSを利用していない。BIOSを利用すると、データ送信後どのくらいの時間でリターンするかの保証がないので、一瞬だけアクティブになる$\overline{\text{BUSY}}$信号を見逃してしまう可能性があるからだ。I/Oポートを直接制御してプリンタにデータを送信するためには、プリンタインターフェイスの8255AポートAにDC1を出力し（Table 6参照）、そのあと$\overline{\text{PSTB}}$を一瞬アクティブにする（Table 7参照）。

Table 6　プリンタインターフェイスへのデータ出力

I/O ポート	意　味
0040H	プリンタインターフェイスの 8255A ポート A bit 7〜0　　プリンタインターフェイスへの出力データ

Table 7　プリンタインターフェイスの $\overline{\text{PSTB}}$ 制御（ノーマルモード）

アドレス	意　味	
0046H	プリンタインターフェイスの 8255A　コントロールポート	
	値	意　味
	0EH (00001110B)	プリンタインターフェイスの $\overline{\text{PSTB}}$ をアクティブにする
	0FH (00001111B)	プリンタインターフェイスの $\overline{\text{PSTB}}$ をインアクティブにする

$\overline{\text{PSTB}}$ はプリンタへのデータストローブ信号で、プリンタは $\overline{\text{PSTB}}$ が LOW レベル（アクティブ）から HIGH レベル（インアクティブ）に変わるときデータを読み込む。$\overline{\text{PSTB}}$ はパラレル信号線に値を出力したあと 1μ 秒以上たってからアクティブにし、さらに 1μ 秒以上たってからインアクティブにしなければならない。この条件を満たすために I/O 命令の間にウェイトを挿入している。これは I/O デバイスのリカバリタイムを満たすことも兼ねている。ウェイトには I/O ポート 5FH を利用する（Table 8 参照）。この I/O ポートは PC-H98 でサポートされたものだが、他機種でも同様の目的に使用できる。

Table 8　0.6μ 秒以上のウェイト用ポート

I/O ポート	意　味
5FH	ウェイト用ポート bit7〜0　　任意

プログラムは List 6 のようになる。

List 6　プリンタへの無動作のコードの送信

```
;DC1 を送る（DC1 以外のコードでもプリンタが余計な動作をしなければ良い）
        mov     al,11H          ; プリンタポートに 11H(DC1) をセット
        out     0040H,al
        out     005FH,al        ;0.6 μ秒ウェイト×2
        out     005FH,al
```

```
                mov     al,0EH          ;PSTB#アクティブ
                out     0046H,al
                out     005FH,al        ;0.6 μ秒ウェイト×2
                out     005FH,al
                or      al,01H          ;PSTB#インアクティブ
                out     0046H,al
        ;BUSY#信号が一瞬アクティブになったらプリンタ電源 ON と判断
        ; 一定時間待ってもアクティブにならなければ電源 OFF と判断
                mov     cx,1000H
BusyCheckLoop:  in      al,0042H        ;BUSY#信号チェック (0 でアクティブ)
                test    al,04H          ;
                jz      online
                loop    BusyCheckLoop
                mov     ax,0005H
                ret
online:         mov     ax,0000H
                ret
offline:        mov     ax,0003H
                ret
```

▶ワンポイント

■簡易セントロニクスモードでの戻り値

簡易セントロニクスモードのときは$\overline{\text{BUSY}}$信号のふるまいからプリンタの状態を推定している。このため、条件によってはデータ送信可能状態とオフライン状態の判定を誤ることがある。

ここで作成したプログラムは、プリンタ内のバッファに余裕があるときだけ正しく動作する。プリンタに大量のデータを送信してバッファが満杯になっている状態でこのプログラムを実行すると、$\overline{\text{BUSY}}$信号がアクティブになっているので、オフライン状態と誤判断する。

また、PC-PR系のページプリンタではオフライン状態でもデータの入力を受けつけるので、オフライン状態の検出はできない。

BIOSをフックするタイプのプリンタスプーラが組み込まれているとき、スプーラがプリンタ出力している最中にここで作成したプログラムを実行すると、DC1コードが出力されるタイミングによっては印字が異常になる場合がある。

■PC-H98ノーマルモードのフルセントロニクスモード

PC-H98ではノーマルモードでもフルセントロニクスモードを選択できる機能が追加された。しかし、この機能は要注意である。

ハードウェア関係の注意点としては、フルセントロニクスモードに移行するとプリンタポートの8255Aにはハイレゾモードと同等の機能が割り当てられることがあげられる。このため、I/Oポー

トを直接操作するプログラムはインターフェイスのモードを判断してI/Oポートを操作する必要がある。0000:0501Hのbit3が1、または、0000:0458Hのbit2が1であればフルセントロニクスモードである。

ソフトウェアの問題としては、フルセントロニクスモードに移行するとプリンタBIOS実行後のステータスがハイレゾと同等の内容になることがあげられる。つまり、BIOSを経由してプリンタを制御するときでもインターフェイスモードを意識したプログラミングを必要とするわけだ。本来ならば、このような拡張を行ったとしてもBIOSレベルでの互換性は保っていて欲しいところだ。また、フルセントロニクスモードでも、ハイレゾモードのプリンタBIOSがサポートしている14H～16Hのコマンドはサポートされない。

結果として、ハイレゾのフルセントロニクスモード、ノーマルの簡易セントロニクスモード、ノーマルのフルセントロニクスモードと3種類のモードが混在することになった。全モードに対応するプログラムを作成するときには上記の差異をよく理解しておかなければならない。

■セントロニクスインターフェイスの$\overline{\text{BUSY}}$信号

本節では$\overline{\text{BUSY}}$信号を負論理のように表現しているが、これはプリンタポートの8255Aの$\overline{\text{BUSY}}$信号ビットが負論理であるためだ。セントロニクスインターフェイスのBUSY信号線の電気的な論理は正論理であるが、プログラムのレベルでこのことを考慮する必要はとくにない。

ライブラリ

GetPrnStat

解説　プリンタの状態を調べる。書式はつぎのとおり。

int **GetPrnStat**(void)

戻り値　プリンタの状態を返す。値と意味はつぎのとおり。

値	状　態
0	データ送信可能（オンライン）状態
1	BUSY状態（ハイレゾ、PC-H98のみサポート）
2	タイムアウトで送信不可能だった（ハイレゾ、PC-H98のみサポート）
3	オフライン状態
4	ペーパーエンド（ハイレゾ、PC-H98のみサポート）
5	プリンタ未接続または電源OFF

サンプル　PRNSTAT.C

COLUMN

フルセントロニクスモードの注意点

ハイレゾモードとPC-H98がサポートしているフルセントロニクスプリンタインターフェイスを使用するときは、プリンタの詳細な情報をチェックして印字の可否を判断している。ところが、フルセントロニクスモードでは簡易セントロニクスでは問題にならなかった、プリンタの端子の微妙な違いが問題になってくる。

そのなかでで一番興味深いのが、EPSONプリンタではインターフェイスに現れる信号が若干異なるため、フルセントロニクスモードのBIOS出力ではEPSONプリンタに送信することができないと言う事実だ。

原因はプリンタコネクタの端子番号18に接続されている$\overline{\mathrm{DC+5V}}$信号（プリンタステータスのAL bit 4）である。PC-PRプリンタではここにプリンタからの$\overline{\mathrm{DC+5V}}$が出力されているが、EPSONプリンタでは未使用となっている。EPSONのプリンタでは同様の信号は端子番号35に出力されている。これはIBM-PC系の仕様と合致させているからである。このため、フルセントロニクスインターフェイスのBIOSはEPSONプリンタが接続されていると常に電源OFFと判断して送信を行わない。

フルセントロニクスモードでEPSONプリンタをサポートするには、直接I/O操作を行ってプリンタ出力しなければならない。この場合、Table Aに示すように$\overline{\mathrm{DC+5V}}$信号以外にもPC-PRプリンタと振舞いの異なる信号があるので、細かな制御をするときには注意が必要である。また、PC-PRプリンタでも、シリアルプリンタとページプリンタでは$\overline{\mathrm{BUSY}}$の動作が少し異なっている。

Table A　代表的なプリンタの各状態におけるステータス（AL）

	ビット番号	7	6	5	4	3	2	1	0
未接続		0	1	0	1	0	1	0	1
PC-PRページプリンタ	ONLINE	0	1	1	0	0	1	0	1
	OFFLINE	1	0	1	0	0	1	0	1
	用紙切れ	1	0	0	0	0	1	0	1
	電源OFF	1	0	1	1	0	1	0	1
PC-PRシリアルプリンタ	ONLINE	0	1	1	0	0	1	0	1
	OFFLINE	1	0	1	0	0	0	0	1
	用紙切れ	1	0	0	0	0	0	0	1
	電源OFF	1	0	1	1	0	1	0	1
EPSONシリアルプリンタ	ONLINE	0	1	1	1	0	1	0	1
	OFFLINE	0	0	1	1	0	0	0	1
	用紙切れ	0	0	0	1	0	0	0	1
	電源OFF	1	0	1	1	0	1	0	0

プリンタのソフトウェアによるオンライン化

PC-PR系シリアルプリンタでは、オフライン状態のときに制御用のコードDC1（11H）をプリンタに送れば、ソフトウェア的にプリンタをオンライン状態に切り替えることができる。ところが、オフラインのときは$\overline{\text{BUSY}}$信号がアクティブになっているため、BIOSのコマンドは、プリンタにデータが送れない状態と判断して、プリンタへコードを送信しない。そこで、直接I/OポートにアクセスしてオフラインのプリンタにDC1コードを送出する方法を解説する。

▶ポイント

- プリンタインターフェイスのモードによって$\overline{\text{PSTB}}$信号の制御方法が変わるのでインターフェイスのモードを判別する。
- データを送信するには、プリンタインターフェイスの8255AポートAに送信データ(DC1)を出力し、そのあと$\overline{\text{PSTB}}$信号を一瞬アクティブにする。

▶プログラミングテクニック

1 フルセントロニクスと簡易セントロニクスを判別する

プリンタポートを直接アクセスする場合、プリンタインターフェイスのモードによって$\overline{\text{PSTB}}$信号の制御方法が変わるのでインターフェイスのモードを判別する必要がある。**システム共通域0000:0501Hのbit3が1ならハイレゾモードなのでフルセントロニクスインターフェイスと判断する**（Table 1参照）。

Table 1　ハイレゾモード判別

アドレス	意　味	
0000:0501H	BIOS制御用フラグ（BIOS_FLAG）	
	bit3	ハイレゾモードとノーマルモードの識別ビット
	1	ハイレゾモード（フルセントロニクス）
	0	ノーマルモード（簡易セントロニクスまたはフルセントロニクス）

ノーマルモードのときは、PC-H98のフルセントロニクスモードかどうか**システム共通域0000:0458Hのbit2**をチェックする。ここが1ならフルセントロニクスモードと判断する。それ以外なら簡易セントロニクスモードと判断する（**Table 2**参照）。プログラムは**List 1**のようになる。

Table 2　PC-H98フルセントロニクスモード判別

アドレス	意　味	
0000:0458H	NESA採用機の情報	
	bit2	**ノーマルモード時のフルセントロニクスと簡易セントロニクスモードとの識別**
	1	ノーマルモード時フルセントロニクスモード
	0	ノーマルモード時簡易セントロニクスモード
	bit1	**ノーマルモード時のフルセントロニクスのサポート**
	1	ノーマルモード時フルセントロニクスモードサポートあり
	0	ノーマルモード時フルセントロニクスモードサポートなし

List 1　フルセントロニクス／簡易セントロニクス判別

```
            mov     ax,0000H                ;ES ← 0000（システム共通域）
            mov     es,ax
            mov     ah,04H                  ;PSTB#アクティブコマンド（フルセントロ用）
            test    es:[0501H],BYTE PTR 08H ;ハイレゾ／ノーマル判定
            jz      FullCentro
            test    es:[0458H],BYTE PTR 04H ;PC-H98ノーマルモードのフルセントロ
            jnz     FullCentro
            mov     ah,0EH                  ;PSTB#アクティブコマンド（簡易セントロ用）
FullCentro:
```

処理の手順

1 フルセントロニクスと簡易セントロニクスを判別する

↓

2 $\overline{\text{BUSY}}$を無視してプリンタにDC1コードを出力する

2 BUSYを無視してプリンタにDC1コードを出力する

普通はプリンタにデータを出力する前に$\overline{\text{BUSY}}$をチェックして、これがアクティブならデータを出力しない。しかしこのルーチンはプリンタがオフライン（$\overline{\text{BUSY}}$がアクティブ）のときに実行するので$\overline{\text{BUSY}}$のチェックは行わない。

プリンタにデータを送信するためには、プリンタインターフェイスの8255AのポートAにDC1（11H）を出力したあと（Table 3参照）、$\overline{\text{PSTB}}$を一瞬アクティブにする（Table 4参照）。

Table 3　プリンタインターフェイスへのデータ出力

I/Oポート	意　味
40H	プリンタインターフェイスの8255A ポートA bit 7〜0　　プリンタインターフェイスへの出力データ

Table 4　プリンタインターフェイスの$\overline{\text{PSTB}}$制御

I/Oポート	意　味	
46H	プリンタインターフェイスの8255A コントロールポート	
	値	意　味
	ノーマルモード	
	0EH（00001110B）	プリンタインターフェイスのPSTB#をアクティブにする
	0FH（00001111B）	プリンタインターフェイスのPSTB#をインアクティブにする
	ハイレゾモード	
	04H（00000100B）	プリンタインターフェイスのPSTB#をアクティブにする
	05H（00000101B）	プリンタインターフェイスのPSTB#をインアクティブにする

$\overline{\text{PSTB}}$はプリンタへのデータストローブ信号で、プリンタは$\overline{\text{PSTB}}$がLOWレベル（アクティブ）からHIGH（インアクティブ）に変化するときデータを読み込む。$\overline{\text{PSTB}}$はパラレル信号線に値を出力したあと1μ秒以上たってからアクティブにし、さらに1μ秒以上たってからインアクティブにしなければならない（これはPC-PRシリーズの場合で、EPSONプリンタではそれぞれ0.5μ秒の間隔をあければ良い）。この条件を満たすためにI/O命令の間にウェイトを挿入している。これはI/Oデバイスのリカバリタイムを満たすことも兼ねている。ウェイトにはI/Oポート5FHを利用する。このI/OポートはPC-H98でサポートされたものだが、他機種でも同様の目的に使用することができる。プログラムはList 2のようになる。

List 2 BUSYを無視してプリンタにDC1コード出力

```
mov     al,11H          ; プリンタポートに11H (DC1) をセット
out     0040H,al
out     005FH,al        ;0.6 μ秒ウェイト
out     005FH,al        ; もう一度
mov     al,ah           ;PSTB#をアクティブに
out     0046H,al
out     005FH,al        ;0.6 μ秒ウェイト
out     005FH,al        ; もう一度
or      al,01H          ;PSTB#をインアクティブに
out     0046H,al
```

▶ワンポイント

■各種プリンタの動作の違い

PC-PR系シリアルプリンタはパネル操作等でオフライン状態になっていても、DC1コードを送信してオンライン状態にできる。ただし、用紙切れなどでパネル操作でもオンラインにできないときにはDC1コードでもオンライン状態にできない。

PC-PR系ページプリンタはDC1コードによるオンライン切り換え、DC3 (13H) コードによるオフライン切り換え機能を持っていないので、パネル操作等でオフラインにした場合、ここで作成した関数でオンラインに切り換えることはできない。また、EPSONプリンタでは、DC1/D3コードによるオンライン状態の切り換えと、パネル操作によるオンライン状態の切り換えは独立した機能となっている。パネル操作でオフラインにした場合、パネル操作でオンラインにするしかない。

ライブラリ

PrinterSelect

解説 プリンタを強制的にオンライン状態にする。書式はつぎのとおり。

void **PrinterSelect**(void)

プリンタポート経由で強制的にオンラインにできない機種 (PC-PRのページプリンタ、ESC/Pプリンタなど) もある。

戻り値 なし

サンプル ONLINEP.C

マウスインターフェイスの存在調査

PC-9800 シリーズの一部の機種にはマウスインターフェイスが用意されていないので、マウスインターフェイスに直接アクセスするプログラムは事前にその有無を確認する必要がある。ここでは、このマウスインターフェイスの有無と、マウスの割り込み周期の変更機能の有無を調べる方法を解説する。

なお、マウスがつながっているかどうかを判定するのは不可能である。

▶ポイント

- PC-9801 初代/E/F1/F2 ではマウスインターフェイスはオプション (PC-9871) になっている。
- PC-98LT/HA にはマウスは接続できない。
- その他の機種はマウスインターフェイスを標準装備している。
- PC-9801 初代/E/F/M ではソフトウェアから割り込み周期を変更することができない。これらの機種ではマウスインターフェイスボード上のディップスイッチで割り込み周期を変更する。
- ハイレゾモードでは割り込み周期は固定されている。

▶プログラミングテクニック

1 機種を判別する

最初に機種を判別する。これによってある程度マウスインターフェイスや割り込み周期変更機能の有無を判別できる（Table 1 参照)。

マウスは PC-98LT/HA 以外の全機種に接続できる。ただし、PC-9801 初代/E/F1/F2 ではマウスインターフェイスボードはオプションとなっている。

マウスインターフェイスには定期的に割り込みを発生させる機能が備わっているが、この割り込み周期をソフトウェアから変更できる機種とできない機種がある。ノーマルモードのマウスインターフェイス内蔵機種のほとんどはOUT 命令で変更できるが、PC-9801 初代/E/F/M ではマウ

スインターフェイスボード上のディップスイッチで変更する必要がある。ハイレゾモードでは120Hzに固定されている。

Table 1　マウスインターフェイスの有無

	マウスインターフェイス	割り込み周期変更
PC-9801初代/E/F1/F2	オプション	なし*1
PC-9801F3/M2/M3	あり	なし*1
PC-98LT/HA	なし	なし
ハイレゾモード	あり	なし
その他	あり	あり*2

*1 マウスインターフェイス上のディップスイッチで変更可能
*2 ソフトウェアで変更可能

これらの機種を判別するにはシステム共通域の0000:0500Hと0000:0501Hを調べる(Table 2参照)。Table 1にあるそれぞれの機種では、これらのビットがTable 3のようになるので、それを判別する。プログラムはList 1のようになる。

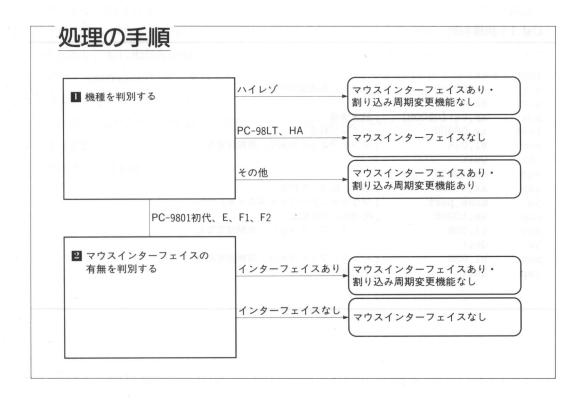



Table 2　機種情報

アドレス	意　味		
0000:0500H	BIOS 制御用フラグ 0（BIOS_FLAG）		
		bit0	機種情報 1（PC-9801 初代/E/F/M、PC-98LT/HA 判別）
		1	その他
		0	PC-9801/E/F/M、PC-98LT/HA、ハイレゾモード
0000:0501H	BIOS 制御用フラグ 1（BIOS_FLAG）		
		bit4	機種情報 3（PC-9801U2、PC-98LT/HA 判別）
		1	PC-9801U2、PC-98LT、PC-98HA
		0	その他
		bit3	機種情報 4（ハイレゾ、ノーマル判別）
		1	ハイレゾモード
		0	ノーマルモード

Table 3　機種判定の方法

機種情報	(1)	(3)	(4)
PC-9801 初代/E/F/M	0	0	0
ハイレゾモード	×	×	1
PC-98LT/HA	0	1	0
上記以外	1	×	0

List 1　機種判別

```
push    es
mov     ax,0000H            ; システム共通域セグメント
mov     es,ax
mov     ax,es:[0500H]       ; 機種情報
test    ax,0800H            ; ハイレゾ判定
mov     bl,01H              ; インターフェイスあり、周期設定なし
jnz     Quit
and     ax,1001H
cmp     ax,0000H            ; 初代、E、F、M 判定
je      mose_port           ; マウスインターフェイスのチェックへ
cmp     ax,1000H            ;PC-98LT、HA 判定
mov     bl,00H              ; インターフェイスなし、周期設定なし
je      Quit
mov     bl,02H              ; インターフェイスあり、周期設定あり
jmp     Quit                ; その他
```

2 マウスインターフェイスの有無を判別する

マウス用8255AのI/Oポートを読み出して、マウス機能の有無を判別する。マウス用8255AはI/Oポートの7FD9H、7FDBH、7FDDH、7FDFHに配置されており、そのなかの**7FD9Hポート**からはマウスボタンの状態などを読み出すことができる（Table 4参照)。この値をチェックすればマウス用8255Aの有無がわかる。

Table 4　マウスインターフェイスの有無

I/Oポート	意　味	
7FD9H	マウスインターフェイスの8255AポートA ボタン押下状態、座標読み取り用ポート（ノーマルモード）	
	ビット	意　味
	bit 7	左ボタン
	bit 6	中ボタン
	bit 5	右ボタン
	bit 4	0
	bit 3〜0	マウスカウンタのデータ（MD3〜MD0)

7FD9Hポートのbit 7〜5はボタンの状態を示し、bit 3〜1はロータリエンコーダのカウンタ入力用である。bit 4は未使用で常に0になっている。このように、このポートから読んだ値は、マウス用8255Aが存在すれば必ず1つ以上のビットが0になっていて、FFHになっていることはない。しかし、もし8255Aが存在しなければCPUが読み出したときにデータバスに信号を出力するデバイスがないので、データバスからはすべてのビットが1であるFFHが読み出される（データバスがプルアップされているため)。

したがって、このポートを読んでFFHではない値が読み出されたらマウスインターフェイスがあると判断できる。

プログラムはList 2のようになる。

List 2　マウスインターフェイスの有無判別

```
        mov     dx,7FD9H        ; ボタン押下状態、座標読み取り用ポート（ノーマル）
        in      al,dx
        cmp     al,0FFH
        mov     bl,00H          ; インターフェイスなし、周期設定なし
        je      Quit
        mov     bl,01H          ; インターフェイスあり、周期設定なし
Quit:   mov     al,bl
        mov     ah,00H
        pop     es
        iret
```

ライブラリ

ExistMouseIF

解説　マウスインターフェイスの存在調査。書式はつぎのとおり。

int **ExistMouseIF**(void)

マウスインターフェイスの有無とその割り込み周期設定機能の有無を調べる。
PC-9871 と F3/M2 内蔵マウスインターフェイスにはソフトウェアから割り込み周期を設定する機能はなく、ボード上の DIP SW で設定する。ハイレゾの割り込み周期は 120Hz で固定になっている。

戻り値　マウスインターフェイスの存在と、割り込み周期機能の有無を返す。

値	意味
0	インターフェイスなし。
1	インターフェイスあり。割り込み周期設定機能なし。
2	インターフェイスあり。割り込み周期設定機能あり。

サンプル　EXISTMIF.C

COLUMN

マウスインターフェイスコネクタとボタン数

PC-9801用のマウスは通常2ボタンマウスであるが、マウスインターフェイスは3ボタンをサポートしている。マウスインターフェイスコネクタの端子番号7は『テクニカルデータブック』ではN.C.（無接続）となっているが、実は中ボタンの入力端子となっている（Table A参照）。中ボタンの状態はI/Oポート7FD9H（マウスインターフェイスの8255AのポートA）のbit 6で読み出すことができる（Table B参照）。

また、PC-9871マウスインターフェイスのマニュアルでは、端子番号1はN.C.となっているが、本体内蔵型と同様＋5Vが出力されている（Table A参照）。マウスインターフェイスコネクタを利用して、別の機器を接続するような場合に有用だろう。

Table A　マウスインターフェイスコネクタの端子接続

I/Oレポート	Read/Write	意　味	
7FD9H（ノーマル） 0061H（ハイレゾ）	Read	PORT Aリード（マウスの状態）	
		ビット	意　味
		bit 7	LEFT　マウスの左ボタンの状態 （1＝OFF、0＝ON）
		bit 6	MID　マウスの中ボタンの状態 （1＝OFF、0＝ON）
		bit 5	RIGHT　マウスの右ボタンの状態 （1＝OFF、0＝ON）
		bit 4	
		bit 3〜0	MD3〜MD0

Table B　マウスインターフェイスのポートA

端子番号	意　味
1	＋5V（マウスは使用していない）
2	ロータリ・エンコーダ端子入力XA
3	ロータリ・エンコーダ端子入力XB
4	ロータリ・エンコーダ端子入力YA
5	ロータリ・エンコーダ端子入力YB
6	左ボタン入力
7	中ボタン入力
8	右ボタン入力
9	GND

サウンド機能の存在調査

サウンド機能は、OPN（FM Operator type-N）と呼ばれるヤマハのFM音源LSI YM2203を中心に構成されている。YM2203は、FM音源部（3音ポリフォニック、4オペレータ、8アルゴリズム）とSSG部（3音ポリフォニック。古典的な音源チップである米GI社製AY-3-8910プログラマブルサウンドジェネレータと互換性がある）からなる音源LSIである。YM2203はI/Oデバイスとして CPUに接続されている。

このFM音源は、すべての機種に搭載されているわけではない。また、このFM音源を簡便にコントロールするサウンドBIOSも、切り離しが可能でEMSメモリとの競合から切り離されていることも多い。そこで、ここでは、サウンド機能を利用するときに事前にサウンド機能やサウンドBIOSを利用できるかどうか確認する方法を解説する。

▶ポイント

- I/Oデバイスが存在しないアドレスからはFFHという値が読みだされることを利用して、サウンド機能の有無を判別する。
- ハイレゾモードのときサウンドBIOS ROMは自動的に切り離されるので、無条件にサウンドBIOSなしと判断する。
- ノーマルモードのときは、ROM領域の所定のアドレスにサウンドBIOSのベクタエントリ情報が存在するかどうかを調べてサウンドBIOSの有無を判断する。

▶プログラミングテクニック

1 FM音源LSIの有無を判別する

サウンド機能を実現しているFM音源LSI（以下OPN）のI/Oポートを読んで、サウンド機能の有無を判別する。

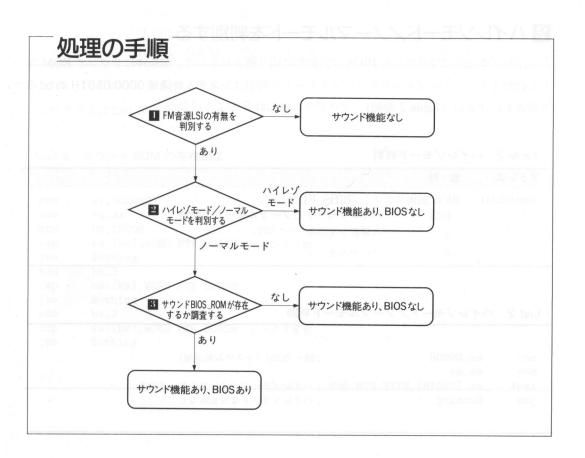
一般的に、I/Oデバイスが存在しないアドレスからはFFHという値が読みだされる。I/Oデバイスが存在しなければ、CPUがI/Oポートを読んだときにデータバスに信号を出力するデバイスはなにもないことになる。データバスはプルアップされているので、こうした状態ではすべての

ビットが1であるFFHが読み出されるのである。

ただし、I/OデバイスがFFHという値を出力する場合や、デバイスの状態によってハイインピーダンスになっている場合もあるので、存在調査はデバイスがFFH以外の値を出力する状態にして行わなければならない。

OPNはI/Oポートの0188H、018AHに配置されている。0188HポートからはOPNのステータスを読み出すことができる（Table 1参照）。

Table 1 FM音源LSIのステータス

I/Oポート	意味	
0188H	FM音源LSIのステータスレジスタ	
	ビット	意味
	bit 7	BUSY（1でFM音源レジスタへの書き込み禁止を示す）
	bit 6～2	未定義
	bit 1	FLAG-A（タイマAのタイムアウトを示す）
	bit 0	FLAG-B（タイマBのタイムアウトを示す）

ステータスのbit 7はBUSYフラグで、1のときFM音源レジスタへの書き込み禁止を表す。このフラグはOPNへの書き込み直後を除いて0になっており、また、1の場合でも一定時間経過すると0になる。bit 6〜2は、メーカーのデータシートでは値が保証されていない。bit 1、0はOPN内蔵タイマの状態で、未使用状態およびタイムアウトで1になる。

したがって、一定時間このポートを読み続けてFFHではない値が読み出されたらOPNが存在すると判断できる。プログラムはList 1のようになる。

List 1　FM音源LSIの有無判別

```
                mov     cx,0100H        ;256回繰り返す
OpnCheckLoop:   mov     dx,0188H        ;OPNステータスポート
                in      al,dx
                cmp     al,0FFH
                jne     OpnExist        ;FFでなければOPNあり
                loop    OpnCheckLoop
                                        ;OPNなし
```

2 ハイレゾモード／ノーマルモードを判別する

ハイレゾモードではサウンドBIOSは自動的に切り離されるので、無条件にサウンドROMなしと判断する。ハイレゾモードとノーマルモードの判別は**システム共通域0000:0501Hのbit 3**を調査すればよい（Table 2参照）。プログラムはList 2のようになる。

Table 2　ハイレゾモード判別

アドレス	意　味	
0000:0501H	BIOS制御フラグ1（BIOS_FLAG）	
	bit3	ハイレゾモードとノーマルモードの判別
	1	ハイレゾモード
	0	ノーマルモード

List 2　ハイレゾモード／ノーマルモード判別

```
mov     ax,0000H                    ;ES ← 0000（システム共通域）
mov     es,ax
test    es:[0501H],BYTE PTR 08H     ;ハイレゾモード／ノーマルモード判定
jnz     Nothing                     ;ハイレゾモードならROMなし
```

3 サウンドBIOS ROMが存在するか調査する

ノーマルモードのときにはサウンドBIOS ROMの存在を調査する。これは、ROM領域の所定のアドレスにサウンドBIOSのベクタエントリ情報が存在するかどうかで判断する。

サウンドBIOS ROMはデフォルトではCC000H〜CFFFFHに配置されている。ROMのオフセット2E00H番地からベクタエントリをセットするための情報が書かれているので、これがTable 3に示したようになっているかをチェックすればROMの有無が判断できる。NEC以外のサウンドBIOSでもここの値は不変である。

Table 3　サウンドBIOS ROMの有無判別

アドレス	意　味	
CC00:2E00H	サウンドBIOSのベクタエントリ情報	
	オフセット	内　容
	2E00H	01H（エントリ数）
	2E01H	00H
	2E02H	00H
	2E03H	00H
	2E04H	D2H（割り込み番号）
	2E05H	00H
	2E06H〜2E07H	割込エントリオフセット

プログラムはList 3のようになる。

List 3　サウンドROMの存在調査

```
mov     ax,0CC00H
mov     es,ax
mov     bx,2E00H                  ;INT ベクタエントリ情報
cmp     es:[bx],WORD PTR 0001H  ; エントリ数
jne     Nothing
add     bx,2
cmp     es:[bx],WORD PTR 0000H
jne     Nothing
add     bx,2
cmp     es:[bx],WORD PTR 00D2H  ; ベクタ番号
jne     Nothing
```

▶ワンポイント

■サウンドROMのアドレス設定

マニュアルには記述がないが、PC-9801-26サウンドボード上のROMはCC00H以外のアドレスにも配置することができる。ROM切り離しのためのジャンパスイッチはFigure 1のような意味を持っている。ただし、ROMアドレスをCC00H以外に設定した場合、既存のプログラムには修正が必要になる。

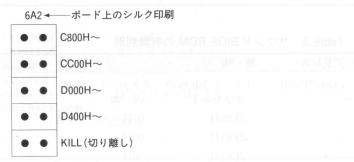



Figure 1　サウンドボード上のジャンパスイッチ

サウンドボード内蔵機種ではほとんどの場合ROMアドレスは移動できない。また、UV11だけはジャンパスイッチでROMの切り離しができない。

ライブラリ

ExistSoundBoard

解説　サウンド機能の有無とそのROMの有無を調べる。書式はつぎのとおり。

int **ExistSoundBoard**(void)

戻り値　サウンド機能とサウンドBIOSの状態。値と意味の関係はつぎのとおり。

値	状　態
0	サウンド機能なし
1	サウンド機能あり。ROMなし（ハイレゾモード含む）
2	サウンド機能あり。サウンドROMあり

サンプル　EXISTSB.C

Appendix

添付ディスクについて

Appendix

添付ディスクの内容

本書には5.25インチおよび3.5インチのフロッピーディスクが添付されています。ディスクにはつぎのものが収録されています。

- 本書で説明した全ライブラリのソースファイル
- C言語から使えるライブラリの .LIBファイル
- ライブラリの関数を使ったサンプルプログラム

ソースファイルは、本文中で説明したテクニックの実例となっています。本文中のプログラムは断片的なものでしたが、添付ディスク中のファイルはそれを組み上げた完成品になっています。これを読んで具体的なプログラムの組みかたを学んでください。

ソースファイルは、C言語から使えるライブラリのソースファイルでもあります。実際にアセンブルやコンパイルして作ったライブラリのSUPER98.LIBファイルもディスクには収録されています。また、関数を使ったサンプルプログラムのソースファイルも収録しました。

添付ディスクは、Figure 1に示すディレクトリ構成になっています。

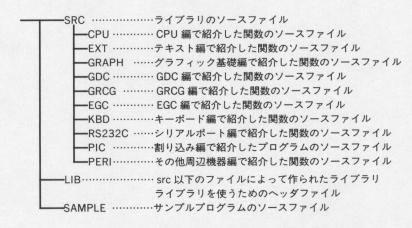



Figure 1　添付ディスクのディレクトリ構成

各編で紹介したライブラリ関数とソースファイル、サンプルプログラムの対応は、Table 1のようになっています。なお、「割り込み編」のプログラムは、その性質上、ライブラリ関数として用意できないので、本文中で説明したテクニックを常駐型プログラムとしてまとめ、サンプルプログラムとして収録しました。

Table 1　ライブラリ関数、ソースファイル、サンプルプログラム

本　文	関　数	ソースファイル	サンプルプログラム
CPU 編	GetSysClk	CPULIB.ASM	CPUTEST.C
	CpuKind	CPULIB.ASM	CPUTEST.C
	CpuReset	CPULIB.ASM	RESET.C
テキスト VRAM 編	GetVram	GETVRAM.ASM	VRAM1.C
	PutVram	PUTVRAM.ASM	VRAM1.C
	PutStrVram	PUTSTRV.C	VRAM2.C
	SetUcgBios	SETUCGBI.ASM	LOADUCG.C
	SetUcgIo	SETUCGIO.ASM	LOADUCG.C
	GetGdcCursor	GETCUR.ASM	VRAM2.C
	SetCursorForm	CURFORM.ASM	CSRFORM.C
	ToSjis	TOSJIS.ASM	JIS2SJIS.C
	ToSjisFast	TOSJISFS.ASM	JIS2SJIS.C
	ToJis	TOJIS.ASM	SJIS2JIS.C
	ToJisFast	TOJISFST.ASM	SJIS2JIS.C
グラフィック基礎編	GraphicInit	GINIT.ASM	GDEMO.C
	VsyncStart	GINIT.ASM	
	PaletteInit	PALETTE.ASM	GDEMO.C、PAL.C
	PaletteAll	PALETTE.ASM	GDEMO.C、PAL.C
	Palette	PALETTE.ASM	
GDC 編	GdcCircle	GDC.ASM	GDEMO.C
	SetPen	GDC.ASM	GDEMO.C
	GdcLine	GDC.ASM	GDEMO.C
	GdcBox	GDC.ASM	GDEMO.C
	GdcScroll	GDC.ASM	GDEMO.C
	KanjiGputc	GPUT.ASM	GDEMO.C
GRCG 編	GraphicBoxf	GRCG.ASM	GDEMO.C
	GraphicCls	GRCG.ASM	GDEMO.C
EGC 編	EgcGraphicBoxf	EGC.ASM	
	EgcKanjiGputc	EGC.ASM	
	GraphicMove	EGC.ASM	GDEMO.C
キーボード編	GetKeyType	KBLIB.ASM	KEYTYPE.C
	GetKbType	KBLIB.ASM	CAPSSW.C、KANASW.C
	GetKeyBeepMode	KBLIB.ASM	KEYBEEP.C
	KeyBeepOn	KBLIB.ASM	KEYBEEP.C
	KeyBeepOff	KBLIB.ASM	KEYBEEP.C
	KeyTouch	KBLIB.ASM	KEYTOUCH.C
	KbSendCommand	KBLIB.ASM	
	KbRecieveData	KBLIB.ASM	
	CapsSwitch	KBLIB.ASM	CAPSSW.C
	KanaSwitch	KBLIB.ASM	
シリアルポート編	RsSendBreak	RSLIB.ASM	
	RsBreakOn	RSLIB.ASM	
	RsBreakOff	RSLIB.ASM	
	SetErOn	RSLIB.ASM	RSSET.C
	SetErOff	RSLIB.ASM	RSSET.C
	SetRsOn	RSLIB.ASM	RSSET.C
	SetRsOff	RSLIB.ASM	RSSET.C
	CheckEr	RSLIB.ASM	RSSTAT.C
	CheckRs	RSLIB.ASM	RSSTAT.C
	CheckCd	RSLIB.ASM	RSSTAT.C
	CheckCs	RSLIB.ASM	RSSTAT.C
	CheckCi	RSLIB.ASM	RSSTAT.C
	CheckDr	RSLIB.ASM	RSSTAT.C
	ReceiveData	RSLIB.ASM	TERM.C
	ReceiveLength	RSLIB.ASM	
	ReceiveSpace	RSLIB.ASM	
	RsOpen	RSLIB.ASM	TERM.C
	RsReOpen	RSLIB.ASM	
	RsClose	RSLIB.ASM	TERM.C

本　文	関　数	ソースファイル	サンプルプログラム
	TransData	RSLIB.ASM	TERM.C
	TransLength	RSLIB.ASM	TERM.C
	TransSpace	RSLIB.ASM	
	GetSpeed	GETSPEED.ASM	RSSTAT.C
	SetSpeed	GETSPEED.ASM	RSSET.C
割り込み編	関数なし	CRTV.ASM	
		KEY_STAT.ASM	
		KEY_VECT.ASM	
		PIC_VECT.ASM	
		TIMER.ASM	
その他周辺機器編	DriveToDaua	DISKLIB.ASM	DISKSTAT.C
	ExistExtRom	PERILIB.ASM	EXISTROM.C
	GetMemorySwitch	PERILIB.ASM	MSW.C
	SetMemorySwitch	PERILIB.ASM	MSW.C
	ExistMouseIF	PERILIB.ASM	EXISTMIF.C
	GetPrnStat	PERILIB.ASM	PRNSTAT.C
	PrinterSelect	PERILIB.ASM	ONLINEP.C
	ExistSoundBoard	PERILIB.ASM	EXISTSB.C

これらのプログラムからライブラリファイルや実行形式ファイルを作ったり、ディスクをコピーしたりするために、Table 2のようなファイルも収録しました。

Table 2　ユーティリティーファイル一覧

種　類	ファイル名	使用アセンブラ	使用コンパイラ
ハードディスクへのインストール用バッチファイル	INST.BAT		
SUPER98.LIBの作成用のバッチファイル	MKLIB_M5.BAT	MASM 5.1	MS-C
	MKLIB_M6.BAT	MASM 6.0	MS-C
	MKLIB_M5.BAT	MASM 5.1	MS-C
	MKLIB_Q6.BAT	MASM 6.0	QuickC
	MKLIB_Q5.BAT	MASM 5.1	QuickC
	MKLIB_B.BAT	TASM	Borland C++
	MKLIB_T.BAT	TASM	Turbo C++
SUPER98.LIB作成用レスポンスファイル	MKLIB.RES		
サンプルのコンパイル用バッチファイル	MKSMP_M.BAT		MS-C
	MKSMP_Q.BAT		Quick C
	MKSMP_B.BAT		Borland C
	MKSMP_T.BAT		Turbo C++
常駐編の実行形式作成用バッチファイル	SRC¥PIC¥MKPIC_M5.BAT	MASM 5.1	
	SRC¥PIC¥MKPIC_M6.BAT	MASM 6.0	
	SRC¥PIC¥MKPIC_T.BAT	TASM	

ディスクの取り扱い

安全のために、添付ディスクは使う前に複製を作り、複製のほうを使うようにしましょう。オリジナルは大切に保管してください。

添付ディスクの複製を作るにはMS-DOSに付属のDISKCOPYコマンドを使います。コピー先ディスクをフォーマットしたあと、オリジナルディスクとコピー先ディスクをそれぞれドライブにセットして(ここでは仮にドライブBとドライブCとします)、つぎのようにコマンドを実行します。ドライブ名を間違えるとオリジナルディスクが破壊されることがありますので、ご注意ください。

```
A>DISKCOPY　B:　C:
```

ハードディスクをお持ちでしたら、できるだけ添付ディスクの内容をハードディスクにコピーしてハードディスクで作業するようにしてください。

ハードディスクにコピーするには、添付ディスクのルートディレクトリでバッチファイルINST.BATを実行します。たとえば、コピー元のフロッピーディスクをドライブB、コピー先のハードディスクをドライブAとしたとき、ハードディスクのディレクトリ¥SUPER98以下にコピーするには、つぎのようにコマンドを実行します。

```
B>INST　A:¥SUPER98
```

これによって、ディレクトリSUPER98が存在しなければ作られ、そのあとで添付ディスクの内容がディレクトリSUPER98以下にコピーされます。

ライブラリの動作環境

ライブラリ関数の対応機種

添付ディスクに収録したライブラリは基本的にノーマルモードの PC-9800 シリーズ全機種に対応していますが、一部動作条件のある関数もあります。関数と、その動作する機種を、Table 1 に示します。

Table 1　ライブラリ関数の動作条件

本　文	関　数	利用条件
CPU 編	GetSysClk	全機種に対応
	CpuKind	全機種に対応
	CpuReset	全機種に対応
テキスト VRAM 編	GetVram	ノーマルモード
	PutVram	ノーマルモード
	PutStrVram	ノーマルモード
	SetUcgBios	ノーマルモード
	SetUcgIo	ノーマルモード
	GetGdcCursor	ノーマルモード
	SetCursorForm	ノーマルモード
	ToSjis	全機種に対応
	ToSjisFast	全機種に対応
	ToJis	全機種に対応
	ToJisFast	全機種に対応
グラフィック基礎編	GraphicInit	ノーマルモード
	VsyncStart	PC-98LT/HA を除く全機種
	PaletteInit	ノーマルモード
	PaletteAll	ノーマルモード
	Palette	ノーマルモード
GDC 編	GdcCircle	ノーマルモード
	SetPen	ノーマルモード
	GdcLine	ノーマルモード
	GdcBox	ノーマルモード
	GdcScroll	ノーマルモード
	KanjiGputc	EGC 搭載機種（EGC タイプ）のみ
GRCG 編	GraphicBoxf	GRCG 搭載機種（VM タイプ、EGC タイプ）のみ
	GraphicCls	GRCG 搭載機種（VM タイプ、EGC タイプ）のみ
EGC 編	EgcGraphicBoxf	EGC 搭載機種（EGC タイプ）のみ
	EgcKanjiGputc	EGC 搭載機種（EGC タイプ）のみ
	GraphicMove	EGC 搭載機種（EGC タイプ）のみ
キーボード編	GetKeyType	全機種に対応
	GetKbType	新キーボード
	GetKeyBeepMode	全機種に対応
	KeyBeepOn	全機種に対応
	KeyBeepOff	全機種に対応
	KeyTouch	全機種に対応
	KbSendCommand	新キーボード

本　文	関　数	利用条件
	KbRecieveData	新キーボード
	CapsSwitch	新キーボード
	KanaSwitch	新キーボード
シリアルポート編	RsSendBreak	全機種に対応
	RsBreakOn	全機種に対応
	RsBreakOff	全機種に対応
	SetErOn	全機種に対応
	SetErOff	全機種に対応
	SetRsOn	全機種に対応
	SetRsOff	全機種に対応
	CheckEr	全機種に対応
	CheckRs	全機種に対応
	CheckCd	全機種に対応
	CheckCs	全機種に対応
	CheckCi	全機種に対応
	CheckDr	全機種に対応
	ReceiveData	全機種に対応
	ReceiveLength	全機種に対応
	ReceiveSpace	全機種に対応
	RsOpen	全機種に対応
	RsReOpen	全機種に対応
	RsClose	全機種に対応
	TransData	全機種に対応
	TransLength	全機種に対応
	TransSpace	全機種に対応
	GetSpeed	全機種に対応
	SetSpeed	全機種に対応
割り込み編	CRTV.COM	他プログラムとの組み合わせによっては動作しない
	KEY_STAT.COM	他プログラムとの組み合わせによっては動作しない
	KEY_VECT.COM	他プログラムとの組み合わせによっては動作しない
	PIC_VECT.COM	他プログラムとの組み合わせによっては動作しない
	TIMER.COM	他プログラムとの組み合わせによっては動作しない
その他周辺機器編	DriveToDaua	全機種に対応
	ExistExtRom	全機種に対応
	GetMemorySwitch	全機種に対応
	SetMemorySwitch	全機種に対応
	ExistMouseIF	全機種に対応
	GetPrnStat	全機種に対応
	PrinterSelect	全機種に対応
	ExistSoundBoard	全機種に対応

必要なソフトウェア

ライブラリを作るには、つぎのソフトウェアが必要です。ハードディスク等にインストールし、環境変数PATHなど必要な設定を済ませておいてください。

■アセンブラ

.ASMファイルをアセンブルして.OBJファイルを作るときに使います。MASM 5.1以上、またはTASM 2.0以上をお使いください。

■Cコンパイラ

.Cファイルをコンパイルして.OBJファイルを作るときに使います。QuickC 2.0、MS-C 6.0、Turbo C++2.0、Borland C++ 2.0での動作が確認されています。

■ライブラリアン

.OBJファイルをまとめて.LIBファイルを作るときに使います。アセンブラやCコンパイラに付属のライブラリアンをお使いください。MASM、MS-C、QuickCにはLIB.EXEが、TASM、Turbo C++、Borland C++にはTLIB.EXEが付属しています。

■リンカ

.OBJファイルをまとめて実行ファイルを作るときに使います。リンカは、アセンブラやCコンパイラに付属したものをお使いください。MASM、MS-C、QuickCにはLINK.EXEが、TASM、Turbo C++、Borland C++にはTLINK.EXEが付属しています。

■その他

MASM 5.1を使用する場合は.COMファイルを作るためにEXE2BIN.EXEが必要となります。EXE2BINは、MS-DOS拡張キットに含まれています。

ライブラリの使い方

ライブラリの作り方

前述したとおり、添付ディスクにはすでにライブラリの.LIBファイルが入っていますが、プログラムを改変した場合や、プログラムから一部だけ抜き出した場合などには、ライブラリを作り直したり、新たにアセンブルしたりする必要があります。

ここでは、ディスクに収録されているライブラリのソースファイルから実際のライブラリを作る方法を説明します。ライブラリを作り直す場合などには、ここで述べる方法を参考にしてください。

ライブラリ作成用としてバッチファイルMKLIB*x*.BATを用意しています。お持ちの処理系に対応するバッチファイルをTable 1に示します。

Table 1　ライブラリ作成用バッチファイル

アセンブラ	コンパイラ	バッチファイル
MASM 5.1	MS-C	MKLIB_M5.BAT
MASM 6.0	MS-C	MKLIB_M6.BAT
MASM 5.1	QuickC	MKLIB_Q5.BAT
MASM 6.0	QuickC	MKLIB_Q6.BAT
TASM	Borland C++	MKLIB_B.BAT
TASM	Turbo C++	MKLIB_T.BAT

ライブラリを作るには、コピーした先の、添付ディスクのルートディレクトリに相当するディレクトリに移動してバッチファイルを実行します。すると、カレントディレクトリに.OBJファイルとSUPER98.LIBが作られます。カレントディレクトリにすでにSUPER98.LIBがある場合、正常に動作しないので、事前に削除しておいてください。

このときに、MASM 5.1には/Mxオプションを、MASM 6.0のML.EXEには/Cxオプションを、TASMには/mxオプションをつけて大文字小文字を区別するように指定してください。TASMにはさらに/jmasm51オプションと/jquirksオプションをつけて、MASM互換の動作をするように指定してください。また、MS-CやQuickCでは、コンパイルのときにスタックチェックを抑止するオプションを指定する必要があります。コマンドラインはたとえばつぎのようになります。

```
TASM  /mx  /jmasm51  /jquirks  SOURCE.ASM
MASM  /Mx  SOURCE.ASM  ;
ML  /c  /Cx  SOURCE.ASM
CL  /Ox  /c  SOURCE.C
QCL  /Ox  /c  SOURCE.C
```

詳しくはライブラリ作成用バッチファイルを見て、アセンブルやコンパイル、ライブラリ作成の手順の参考にしてください。

実際の開発ではMAKEユーティリティを使うほうが作業を効率よく進めることができます。MAKEユーティリティについては、それぞれのマニュアルを参照してください。

なお、ディレクトリSRC¥PICに収録した「割り込み編」のプログラムは、ライブラリ関数ではなく独立した常駐型プログラムなので独立して実行形式を作成する必要があります。実行形式を作成するためにバッチファイルMKPIC*x*.BATを用意しています。お持ちの処理系とバッチファイルの対応をTable 2に示します。ディレクトリSRC¥PICに移動してこのバッチファイルを実行すると、カレントディレクトリに.OBJファイルと実行形式が作られます。

Table 2　割り込み編の実行形式作成用バッチファイル

アセンブラ	バッチファイル
MASM 5.1	SRC¥PIC¥MKPIC_M5.BAT
MASM 6.0	SRC¥PIC¥MKPIC_M6.BAT
TASM 2.0	SRC¥PIC¥MKPIC_T.BAT

ライブラリの利用法

ライブラリの関数を呼び出すCのソースファイルでは、つぎのようにヘッダファイルSUPER98.Hをインクルードしてください。

```
#include "super98.h"
```

そして、リンクするときにライブラリSUPER98.LIBを指定します。コンパイルするときには、カレントディレクトリにSUPER98.Hを置くか、/Iオプション（−Iオプション）でSUPER98.Hファイルのあるディレクトリを指定する必要があります。コマンドラインは、たとえばつぎのようになります。

```
CL   /ILIB   SOURCE.C   LIB¥SUPER98.LIB
QCL  /ILIB   SOURCE.C   LIB¥SUPER98.LIB
TCC  −ILIB   SOURCE.C   LIB¥SUPER98.LIB
BCC  −Ilib   SOURCE.C   LIB¥SUPER98.LIB
```

インクルードと指定の方法については、サンプルプログラムとバッチファイルを参考にしてください。

サンプルプログラムの使い方

サンプルプログラムの作り方

添付ディスクには、Table 1 のサンプルプログラムを収録してあります。

Table 1　サンプルプログラム

サンプルプログラム	使っているライブラリ関数	内　容
CPUTEST.C	GetSysClk、CpuKind	CPU の種類とシステムクロック周波数を取得する
VRAM1.C	GetVram、PutVram	画面上のテキスト画面を左右反転する
VRAM2.C	PutStrVram、GetGdcCursor	文字列を入力して、VRAM 書き込みによりエコーバックする
CSRFORM.C	SetCursorForm	カーソルの形を変更する
LOADUCG.C	SetUcgBios、SetUcgIo	ユーザー定義文字を登録する
JIS2SJIS.C	ToSjis、ToSjisFast	JIS をシフト JIS に変換する
SJIS2JIS.C	ToJis、ToJisFast	シフト JIS を JIS に変換する
GDEMO.C	GraphicInit、PaletteInit、PaletteAll、SetPen、GdcCircle、GdcLine、GdcBox、GdcScroll、KanjiGputc、GraphicBoxf、GraphicCls	グラフィックのサンプル
PAL.C	PaletteInit、Palette	パレットを変更する
KEYTYPE.C	GetKeyType	キーボードの種別を取得する
KEYBEEP.C	GetKeyBeepMode、KeyBeepOn、KeyBeepOff	キーバッファフロー時のビープを ON または OFF する
KEYTOUCH.C	KeyTouch	キーの連続押下状態を取得する
CAPSSW.C	CapsSwitch、GetKbType	CAPS キーのロック状態を制御する
KANASW.C	KanaSwitch、GetKbType	カナキーのロック状態を制御する
TERM.C	RsOpen、TransData、ReceiveLength、ReceiveData、RsClose	簡単な通信ソフト
RSSET.C	SetErOn、SetErOff、SetRsOn、SetRsOff、SetSpeed	シリアルポートの信号線と通信速度を設定する
RSSTAT.C	CheckEr、CheckRs、CheckCd、CheckCs、CheckCi、CheckDr、GetSpeed	シリアルポートの信号線の状態と通信速度を取得する
DISKSTAT.C	DriveToDaua	ドライブの種類を取得する
EXISTROM.C	ExistExtRom	指定したアドレスに ROM があるか調べる
MSW.C	GetMemorySwitch、SetMemorySwitch	メモリスイッチを調べたり、設定したりする
EXISTMIF.C	ExistMouseIF	マウスインターフェイスがあるか調べる
PRNSTAT.C	GetPrnStat	プリンタの状態を取得する
ONLINEP.C	PrinterSelect	プリンタをオンラインにする
EXISTSB.C	ExistSoundBoard	サウンド機能があるか調べる

サンプルプログラムをコンパイルして実行形式を得るには、コピーした先の、添付ディスクのルートディレクトリに相当するディレクトリに移動してサンプルプログラム作成用バッチファイル MKSMPx.BAT を実行してください。これによってコンパイルが行なわれ実行ファイルがすべて作られます。処理系とバッチファイルの対応を Table 2 に示します。

Table 2　サンプルプログラム作成用バッチファイル

アセンブラ	バッチファイル
MASM 5.1	SRC¥PIC¥MKPIC_M5.BAT
MASM 6.0	SRC¥PIC¥MKPIC_M6.BAT
TASM 2.0	SRC¥PIC¥MKPIC_T.BAT

自分のプログラムのコンパイルやリンクのためにメイクファイルを作るときには、このバッチファイルを参考にしてコンパイルオプションを与えてください。

サンプルプログラムの実行上の注意

これらのプログラムはハードウェアを直接操作しており、その性格上デバイスドライバ（EMS ドライバなど）や常駐ソフトなどの他のソフトウェアとの競合によりプログラムの正常な動作が行なわれない場合や、他のソフトウェアの動作に対して悪影響を与える場合があります。

本文で紹介できなかった関数

添付ディスクのライブラリのなかには、本文では紹介できなかった関数がいくつか収録されています。

漢字コードの変換

ToSjis

解説　JIS 漢字コードからシフト JIS 漢字コードへの変換。書式はつぎのとおり。

unsigned int **ToSjis**(unsigned int *code*)

code　16 ビット値のうち、上位バイトに JIS 漢字コードの 1 バイト目を、下位バイトに JIS 漢字コードの 2 バイト目を入れる。

戻り値　16ビット値のうち、上位バイトが変換後のシフトJIS漢字コードの1バイト目で、下位バイトが変換後のシフトJIS漢字コードの2バイト目である。引数の値がシフトJIS漢字コードに変換できない無効な場合は0000Hを返す。

サンプル　SJIS2JIS.C

ToSjisFast

解説　JIS 漢字コードからシフト JIS 漢字コードへの変換。書式はつぎのとおり。

unsigned int **ToSjisFast**(unsigned int *code*)

code　16 ビット値のうち、上位バイトに JIS 漢字コードの 1 バイト目を、下位バイトに JIS 漢字コードの 2 バイト目を入れる。

漢字コードの有効性のチェックをしないぶんだけ ToSjis 関数より高速である。

戻り値　16 ビット値のうち、上位バイトが変換後のシフト JIS 漢字コードの 1 バイト目で、下位バイトが変換後のシフト JIS 漢字コードの 2 バイト目である。引数の値がシフト JIS 漢字コードに変換できない無効な場合は戻り値は不定となる。

サンプル　SJIS2JIS.C

▶▶ CONTINUED

ToJis

解説　シフト JIS 漢字コードから JIS 漢字コードへの変換処理。書式はつぎのとおり。

unsigned int **ToJis**(unsigned int *code*)

code　16 ビット値のうち、上位バイトにシフト JIS 漢字コードの 1 バイト目を、下位バイトに 2 バイト目を入れる。

戻り値　16 ビット値のうち、上位バイトが変換後の JIS 漢字コードの 1 バイト目、下位バイトが変換後の JIS 漢字コードの 2 バイト目になる。引数が JIS 漢字コードに変換できない無効なコードの場合は 0000H を返す。

サンプル　JIS2SJIS.C

ToJisFast

解説　シフト JIS 漢字コードから JIS 漢字コードへの変換処理の高速版。書式はつぎのとおり。

unsigned int **ToJisFast**(unsigned int *code*)

code　16 ビット値のうち、上位バイトにシフト JIS 漢字コードの 1 バイト目を、下位バイトに 2 バイト目を入れる。JIS 漢字コードに変換できない無効なコードを指定してはならない。

漢字コードの有効性のチェックをしないぶんだけToJis関数より高速である。

戻り値　16 ビット値のうち、上位バイトが変換後の JIS 漢字コードの 1 バイト目、下位バイトが変換後の JIS 漢字コードの 2 バイト目になる。

サンプル　JIS2SJIS.C

EGCによるグラフィックのブロック移動

GraphicMove

解説　四角形の指定領域のグラフィック画面の内容を指定領域に移動する。書式はつぎのとおり。

void **GraphicMove**(int *X1*,int *Y1*,int *X2*,int *Y2*,int *X3*,int *Y3*)

X1, *Y1*　転送元の左上の点の座標
X2, *Y2*　転送元の右下の点の座標
X3, *Y3*　転送先の左上の点の座標

EGCの4プレーン同時ブロック転送とビット単位のシフト機能を利用して、指定の領域のグラフィック画面の内容を別の位置に高速に転送する。EGCタイプのPC-9801でのみ動作する。

戻り値　なし

サンプル　GDEMO.C

ToJis

解説　シフトJIS漢字コードからJIS漢字コードへの変換処理。書式はつぎのとおり。

unsigned int ToJis(unsigned int *code*)

ToJisFast

unsigned int ToJisFast(unsigned int *code*)

資料集

資料集の表記について

- 資料集では機種名から“PC–9801”あるいは“PC–98”を略して表記する。機種名を併記する場合は、“・”でつなげて表記する。
- また、ノーマルモードはLT・HAとH98を除く640×400ドットの画面をもつ動作モードを指し、ハイレゾモードとはH98シリーズを除く1120×750ドットの画面をもつ動作モードのことである。
- 特定の機種や動作モード、CPUに依存する項目は、その項目の最後に“[”、“]”で囲んで、その機種やモードを示す。
- 資料の中には独自の解析結果も含まれている。解析結果には2つのレベルを設けた。1つは、解析結果ではあるが多くの機種で確認されており、ほぼ確定しているもの。もう1つは、多くの機種で確認されておらず、あくまでも参考とするにとどめておいてもらいたいものである。確認された項目にには†を、未確認の項目には‡をつけた。
- 公開されている資料に関しても、資料に誤りがあると思われる部分は指摘を入れた。

システムの管理するメモリ一覧

割り込みベクタ
0000:0000〜0000:03FFH

0000:0000〜0000:03FFHのメモリは、割り込み（ハードウェア、ソフトウェアの両方を含む）に対応するハンドラのアドレスを格納したテーブルである。割り込み番号に対応するアドレスをつぎに示す。

アドレス	+00	+04	+08	+0C	+10	+14	+18	+1C
0000H	00	01	02	03	04	05	06	07
0020H	08	09	0A	0B	0C	0D	0E	0F
0040H	10	11	12	13	14	15	16	17
0060H	18	19	1A	1B	1C	1D	1E	1F
0080H	20	21	22	23	24	25	26	27
00A0H	28	29	2A	2B	2C	2D	2E	2F
00C0H	30	31	32	33	34	35	36	37
00E0H	38	39	3A	3B	3C	3D	3E	3F
0100H	40	41	42	43	44	45	46	47
0120H	48	49	4A	4B	4C	4D	4E	4F
0140H	50	51	52	53	54	55	56	57
0160H	58	59	5A	5B	5C	5D	5E	5F
0180H	60	61	62	63	64	65	66	67
01A0H	68	69	6A	6B	6C	6D	6E	6F
01C0H	70	71	72	73	74	75	76	77
01E0H	78	79	7A	7B	7C	7D	7E	7F
0200H	80	81	82	83	84	85	86	87
0220H	88	89	8A	8B	8C	8D	8E	8F
0240H	90	91	92	93	94	95	96	97
0260H	98	99	9A	9B	9C	9D	9E	9F
0280H	A0	A1	A2	A3	A4	A5	A6	A7
02A0H	A8	A9	AA	AB	AC	AD	AE	AF
02C0H	B0	B1	B2	B3	B4	B5	B6	B7
02E0H	B8	B9	BA	BB	BC	BD	BE	BF
0300H	C0	C1	C2	C3	C4	C5	C6	C7
0320H	C8	C9	CA	CB	CC	CD	CE	CF
0340H	D0	D1	D2	D3	D4	D5	D6	D7
0360H	D8	D9	DA	DB	DC	DD	DE	DF
0380H	E0	E1	E2	E3	E4	E5	E6	E7
03A0H	E8	E9	EA	EB	EC	ED	EE	EF
03C0H	F0	F1	F2	F3	F4	F5	F6	F7
03E0H	F8	F9	FA	FB	FC	FD	FE	FF

システム共通域
0000:0400〜0000:05FFH

0000:0400〜0000:05FFHの間は、PC–9801の諸設定の保存やBIOSのワークエリアとして利用され、システム共通域と呼ばれる。システム共通域の値のなかには、CPUやI/Oの状態など現在のPC–9801の動作環境が示されている。

アドレス	意味
0400H	BIOSフラグ2（BIOS_FLAG2） ビット　意味 bit7　98NOTE識別 （1→98NOTE／0→その他） bit5　レジュームON/OFF［HA・NV・NS/E・NC］† （1→レジュームON／0→レジュームOFF） HANDY/NOTEメニューのレジューム機能の使用する/使用しないに従ってセットされる。ここを直接書き換えてもメニューのON/OFF状態は変化しない。直後の電源OFFおよびCTRL+GRPH+HELPではメニューの設定に関りなくこのビットの示す動作をする。電源ONするとメニューのON/OFF状態がコピーされる。 bit3　V33A識別 （1→V33A／0→その他） bit2　動作クロック‡ （1→80386 16MHz［RA21］/20MHz［DA2］／0→80386 20MHz［RA21］/16MHz［DA2］） bit1　動作CPU‡ （1→80386／0→その他・80386のV30相当モード）［RA21・DA2］ （BIOSで1にセット）［HA］ bit0　B4670/4680インターフェイスの有無‡ （1→あり／0→なし）
0401H	使用可能プロテクトメモリ容量（EXPMMSZ）† 128Kバイト単位で使用可能なプロテクトメモリの容量を示す。RAMDISK.SYS、EMM.SYSなどの資源管理用。プロテクトメモリを使用するときは、必要とするメモリ量に相当するだけこの値を減らす。高位のアドレスからの必要量を使うことができる。
0402H	起動時のセレクタ番号（SYS_SEL）［ハイレゾ］† ハイレゾモードの起動用セレクタ番号が格納されている。
0403H	ITF用ワークエリア（ITF_WORK）‡
0404〜0407H	シャットダウン時のスタックポインタ保存位置† 80286以降のCPU搭載機では、SHUT 0が0のときシャットダウンするとITFはSP←0404H、SS←0406HとセットしたのちRETFする。
0408〜040FH	シフトキーコードテーブル（SB_SHIFT_CODE）［ハイレゾ］ 最大8個のシフトキーを登録できる。登録されているキーが押されると、KB_SHFT_STS（053AH）の対応するビットが1になる。対応は以下のとおり。 アドレス　KB_SHFT_STS　起動時の値 0408H　bit7　FFH 0409H　bit6　FFH 040AH　bit5　FFH 040BH　bit4　74H（CTRL） 040CH　bit3　73H（GRPH） 040DH　bit2　72H（カナ） 040EH　bit1　71H（CAPS） 040FH　bit0　70H（SHIFT）
0410〜0417H	EMSレジューム情報保存［HA］ レジュームのため、EMSカード制御用I/Oポートに出力する内容を保存する。 アドレス　意味 0410H　ポート08E9H（C000H SEL PKG）に出力するデータ 0411H　ポート08E1H（C000H SEL BANK）に出力するデータ 0412H　ポート08E9H（C400H SEL PKG）に出力するデータ

アドレス	意味
	0413H　ポート 08E3H（C400H SEL BANK）に出力するデータ 0414H　ポート 08E9H（C800H SEL PKG）に出力するデータ 0415H　ポート 08E5H（C800H SEL BANK）に出力するデータ 0416H　ポート 08E9H（CC00H SEL PKG）に出力するデータ 0417H　ポート 08E7H（CC00H SEL BANK）に出力するデータ
0410～ 0413H	キーバッファ開始アドレス（KB_BUF_ADR）［ハイレゾ］ 起動時は 0000:0502H
0414～ 0417H	キーボードのコード変換テーブルアドレス（KB_ENTRY_TBL_ADR）［ハイレゾ］
0418～ 041BH	内部割り込みキー設定テーブルアドレス（KB_INT_ADR）［ハイレゾ］
041C～ 041DH	プリンタのタイムアウト値（PR_TIME）［ハイレゾ］ 10 ミリ秒単位でタイムアウト値を設定する。0 を設定するとタイムアウトしない。
041E～ 0443H	仮想ディスク関連パラメータテーブル‡

アドレス	名称	意味
041EH	VD_CMD	リクエストコード
041FH	VD_SEC_UNIT	セクタ数
0420H	VD_DRIVE	仮想ドライブ番号
0421H	VD_TRACK	トラック番号
0422H	VD_SEC	セクタ番号
0423H	VD_NUL	ヌルデータ
0424～0425H	VD_REMAIN_SEC	残りセクタ数
0426～0427H	VD_REMAIN_LEN	残りデータ数
0428～0429H	VD_DATA_OFF	データのオフセットアドレス
042AH	VDISK_EQUIP	仮想ディスクのマウント状態
042BH	BRANCH_INT	BRANCH 4670/4680 インターフェイスの外部割り込み番号
042C～042FH		BRANCH BIOS のワークエリアアドレス
0430H	VD_BOOT_WORK	仮想ディスクから起動する時のワークエリア
0440H	VD_ADD	DISK BIOS から仮想ディスク BIOS へのエントリアドレス。INT 1BH から仮想ディスク BIOS へディスパッチする

アドレス	意味
0444～ 0447H	マルチイベントタイマのパラメータアドレス（CAL_ROOT_LIST）［ハイレゾ］ マルチイベントタイマの待ち行列のパラメータブロックへのポインタ
0448～ 0449H	BEEP の継続時間カウンタ（CAL_BEEP_TIME）［ハイレゾ］ AH＝24H、INT 18H の BEEP の残り時間を 10 ミリ秒単位で格納する
044A～ 044BH	BEEP 音程（CAL_TONE）［ハイレゾ］ AH＝23H、INT 18H で設定する音程。8253A の分周値が格納される。
044C～ 044FH	シングルイベントタイマのタイムアウトアドレス（CAL_USER_OFF/SEG）［ハイレゾ］ シングルイベントタイマがタイムアウトしたときに呼び出すユーザルーチンのアドレス
0455H	名称不明 bit8　意味 1　システムセットアップメニュー表示後起動‡ 0　リセット起動（DA で確認。H98S は無変化）
0456H	レジューム用 POWER OFF 禁止フラグ［NOTE・HA］† レジューム ON のとき、電源 OFF してはならないタイミングで 0 以外になる。BIOS がセットする。
0457H	98NOTE 機種識別［NOTE］† 値　意味 00H　N・NV 3FH　NS・NS/E・NC・NS/T フロッピーディスクモデル 27H　NS・NS/E 20M バイトハードディスクモデル 256 バイトセクタ A7H　NS・NS/E 20M バイトハードディスクモデル 512 バイトセクタ 37H　NS/E・NC 40M バイトハードディスクモデル 256 バイトセクタ B7H　NS/E・NC 40M バイトハードディスクモデル 512 バイトセクタ 2FH　NS/T 80M バイトハードディスクモデル 256 バイトセクタ AFH　NS/T 80M バイトハードディスクモデル 512 バイトセクタ
0458H	NESA 情報（1） ビット　意味 bit7　NESA 識別 （1→NESA［H98］／0→非 NESA） bit3　H98S 動作速度‡ 1→80486 20MHz 0→80386 20MHz/8MHz bit2　プリンタインターフェイスモード［H98 ノーマル］ 1→フルセントロニクスモード 0→簡易セントロニクスモード bit1　プリンタインターフェイスのタイプ［H98 ノーマル］ 1→フルセントロニクス利用可 0→フルセントロニクス利用不可
0459H	NESA 情報（2）［H98］ ビット　意味 bit3　ディスプレイ種別［H98 ハイレゾ］ 1→ノンインターレース 0→インターレース
0460～ 047FH	SCSI 接続ドライブの諸元† 4 バイト 1 組で ID ごとに 8 ドライブ分格納されている。接続されていないドライブはすべて 0。システム起動時と BIOS イニシャライズ時にセットされる。

アドレス	対応するデバイス
0460～0463H	SCSI#0
0464～0467H	SCSI#1
0468～046BH	SCSI#2
046C～046FH	SCSI#3
0470～0473H	SCSI#4
0474～0477H	SCSI#5
0478～047BH	SCSI#6
047C～047FH	（ID#7 のドライブは認識されない）

各 4 バイトの意味は以下のとおり。
0 バイト目←セクタ数
1 バイト目←ヘッド数
2～3 バイト目←セクタ種別とセクタ長、シリンダ数

ビット	意味
bit15	セクタ種別（1→ハードセクタ／0→ソフトセクタ）
bit14	―――
bit13～12	セクタ長

アドレス	意味

値　意味
10B　1024 バイト
01B　512 バイト
00B　256 バイト
bit11～0　シリンダ数

0480H　システムタイプ（SYS_TYPE）

ビット	意味
bit7	新センス（AH = 84H、INT 1BH）サポートの SASI インターフェイスの有無†（1→あり／0→なし）
bit4	フロッピーディスクモータ制御（1→自動停止モード／0→常時 ON）
bit3	最後にアクセスしたドライブのタイプ† 1→両用フロッピーディスクドライブ 0→1M バイト専用外付フロッピーディスクドライブ
bit1～0	拡張 CPU タイプ

値	意味
11B	80386・80486（除く H98 の 80386 8MHz モード）
01B	80286
00B	8086・V30・V30HL・V50・H98 の 80386 8MHz モード

0481H　キーボードタイプ（KEYB_TYPE）

ビット	意味
bit7	未使用
bit6、3	キーボード識別（1）、（2） 新キーボードを持つ本体に旧キーボードを接続したとき、起動時にキーボードが接続されていなかったとき、および旧キーボードを持つ本体に新キーボードを接続したときも 00B。

値	意味
11B	新キーボード（NUM キーあり、DIP SW 2-7 OFF）
01B	新キーボード（NUM キーあり、DIP SW 2-7 ON）
10B	新キーボード（NUM キーなし）
00B	旧キーボード（全種類）

ビット	意味
bit5	NEC B000 バンクページフレーム用メモリ†［ノーマル］（1→あり／0→なし）
bit4	バンク型 EMS 使用時のバンク状態（EMM.SYS 使用時）†［ノーマル］ 1→ページフレーム 0→グラフィック VRAM
bit3	キーボード識別（2） bit6 参照
bit2	1→RL（ハイレゾ）・H98（ハイレゾ） 0→RL（ノーマル）・H98（ノーマル）
bit1	SASI#1 パラメータポインタ（05E8H）関連‡
bit0	SASI#0 パラメータポインタ（05ECH）関連‡

0482H　SCSI ハードディスク接続状態（DISK_EQUIPS）

ビット	意味
bit7	
bit6	SCSI#6
bit5	SCSI#5
bit4	SCSI#4
bit3	SCSI#3
bit2	SCSI#2
bit1	SCSI#1
bit0	SCSI#0

（1→あり／0→なし）

0483H　SCSI インターフェイスのリザルト‡

0484H　CPU タイプ（CPU_TYPE）

値	意味
bit7～6	内蔵ハードディスクの DMA チャネル（除 NESA）

11B→チャネル 3
10B→チャネル 2
01B→チャネル 1
00B→チャネル 0

bit5～4　内蔵ハードディスクの種類（除 NESA）‡

値	意味
11B	SCSI 内蔵ハードディスクドライブあり
10B	SCSI 内蔵ハードディスクドライブあり（DA7）
01B	RL-SASI 内蔵ハードディスクドライブあり
00B	SCSI/RL-SASI 内蔵ハードディスクドライブなし

bit3～0　CPU クロック判別

条件	bit3～0	CPU クロック
80386 80486	1111B	80386SL 20MHz
	1110B	80486SX 16MHz
	0011B	80386SX 20MHz
	0010B	80386DX 20MHz
	0010B	V30 10MHz 相当
	0001B	80386SX 12MHz
	0011B	80386SX 16MHz
	0001B	80386DX 16MHz 相当
	0000B	80386DX 16MHz
	0000B	80386DX 16MHz 相当
	0000B	V30 8MHz 相当
80286	*xx*1*x*B	80286 12MHz
	*xx*0*x*B	80286 12MHz 以外
NESA 対応機	000*x*B	80386 8MHz 相当
	001*x*B	80386 20MHz 相当
	010*x*B	80386 25MHz
	011*x*B	80386 33MHz 相当
	100*x*B	80486 25MHz
	101*x*B	80486 20MHz
V30	0000B	V30 8/10MHz
V30HL	000*x*B	V30HL 8MHz
V33A	001*x*B	V33A 16MHz

0485H　DISK BIOS の 2DD 用ワークエリア‡

0486～0487H　CPU リセット時の DX レジスタの値†
RA 以降の 80386・80486 機で有効
（機種によって V30 相当モードではセットされない）
80386DX = 0308H［RA21・DA2・NS］/0305H［RA2］
80386SX = 2308H［NC・T・NS/E］/2305H［ES・RS・LS］
80486DX = 0404H
80486SX = 0420H［FA・H98-90］
487SX = 0421H
NS/T は 08F2H ポートの内容

0488H　RAM ドライブ接続状態（RDISK_EQUIP）

ビット	意味
bit7	640K バイトフロッピーディスクユニット#3
bit6	640K バイトフロッピーディスクユニット#2
bit5	640K バイトフロッピーディスクユニット#1
bit4	640K バイトフロッピーディスクユニット#0
bit3	1M バイトフロッピーディスクユニット#3
bit2	1M バイトフロッピーディスクユニット#2
bit1	1M バイトフロッピーディスクユニット#1
bit0	1M バイトフロッピーディスクユニット#0

（1→RAM ドライブ／0→非 RAM ドライブまたはドライブなし）RAM ドライブが存在するとき、同時に DISK_EQUIP（055CH）の同じインターフェイスタイプ、ドライブを示す装置番号のビットも 1 になる。

0489～048DH　RAM ドライブ BIOS ワーク‡
RAM DRIVE BIOS から ROM BIOS 内の DISK BIOS エントリに FAR ジャンプするためのコード

0492H　両用フロッピーディスクリキャリブレイト要求フラグ（DISK_RESET）

ビット	意味
bit7	640K バイトフロッピーディスクユニット#3
bit6	640K バイトフロッピーディスクユニット#2
bit5	640K バイトフロッピーディスクユニット#1
bit4	640K バイトフロッピーディスクユニット#0

アドレス	意　味
	bit3　1Mバイトフロッピーディスクユニット#3 bit2　1Mバイトフロッピーディスクユニット#2 bit1　1Mバイトフロッピーディスクユニット#1 bit0　1Mバイトフロッピーディスクユニット#0 (1→リキャリブレイト要求あり／0→なし)
0493H	両用フロッピーディスク、1Mバイトインターフェイスモード時のアクセスモード(F2HD_MODE) 両用インターフェイスが1Mバイトインターフェイスモードのとき、接続されているドライブのアクセスモード/(初期値 = FFH(倍密/両面))。倍密、単密指定(1→倍密(96tpi)／0→単密(48tpi)) ビット　意　味 bit7　ユニット#3 bit6　ユニット#2 bit5　ユニット#1 bit4　ユニット#0 両面、片面指定(1→両面／0→片面)
0494H	両用フロッピーディスク、640Kバイトモード時のドライブ接続状態(DISK_EQUIP2) (実機で調査したところ、テクニカルデータブックの記載と異なり、1Mバイト フロッピーディスクを接続してもこのアドレスの相応するビットが1にならなかった)
0495H	GRCG/EGCモードレジスタ設定値(GRAPH_CHG) GRCG/EGCのモードレジスタの設定値。アプリケーションはGRCG/EGCに書き込みを行ったときには、必ず同時にこのアドレスにも書き込みを行わなければならない。マウスドライバが割り込みルーチン内でGRCG/EGCを使用したとき、このアドレスの内容を参照してGRCG/EGCのレジスタ内容を復帰するために必要だからである。
0496～ 0499H	GRCG/EGCタイルレジスタ設定値(GRAPH_TAL) GRCG/EGCのタイルレジスタ#0、#1、#2、#3の設定値。GRCG/EGCモードレジスタ設定値(GRAPH_CHG)と同様に使用する。
04AC～ 04AFH	DISK BIOSのジャンプ先アドレス格納用ワークエリア(XROM_PTR) 本体BIOS内のINT　1BHルーチンから拡張ROM BIOSにジャンプするとき、ジャンプ先アドレスを格納する。
04B0～ 04BFH	拡張DISK BIOSテーブル(DISK_XROM) 拡張DISK BIOSのROMアドレスを示すテーブル。DA = 00～0FHのそれぞれに拡張DISK BIOSのアドレス上位8ビットを格納する。0なら本体内DISK BIOSによるサポートまたはサポートなし。拡張ROMのINIT 2(000FHエントリ)でセットする。 アドレス　意　味 04B0H　DA = 00H SASIハードディスク(リニアセクタ番号指定) 04B1H　DA = 01H 両用フロッピーディスク1Mバイトインターフェイスモードの640Kバイトアクセスモード 04B2H　DA = 02H SCSIハードディスク(リニアセクタ番号指定) 04B3H　DA = 03H 04B4H　DA = 04H 04B5H　DA = 05H 320Kバイトフロッピーディスク 04B6H　DA = 06H 04B7H　DA = 07H 640Kバイトフロッピーディスク、両用フロッピーディスク640Kバイトインタフェイスモードの640Kバイトアクセスモード 04B8H　DA = 08H SASIハードディスク(絶対セクタ番号指定) 04B9H　DA = 09H 1Mバイトフロッピーディスク、両用フロッピーディスク1Mバイトインターフェイスモードの1Mバイトアクセスモード 04BAH　DA = 0AH SCSIハードディスク(絶対セクタ番号指定) 04BBH　DA = 0BH 04BCH　DA = 0CH SCSIデバイス 04BDH　DA = 0DH 04BEH　DA = 0EH 04BFH　DA = 0FH 両用フロッピーディスク640Kバイトインターフェイスモードの1Mバイトアクセスモード
04D0～ 04F3H	拡張ROM IDテーブル(XROM_ID) ノーマルモードは04D0～04DFH、ハイレゾモードは04E0～04F3Hを使う。拡張ROMが、ROM領域占有を宣言するために固有の値を格納する。拡張ROMのINIT 1(000CHエントリ)でセットする。 ビット　意　味 bit7　拡張ROMの有無 (1→あり／0→なし) bit6　ジャンプテーブルの有無 (1→あり／0→なし)　8Kバイト以上の拡張ROMの4Kバイト以降の領域の場合0 bit5～0　拡張ROM識別コード 拡張ROM毎に固有な00～3FHの値 システム共通域アドレスと対応する拡張ROMのアドレスはつぎのとおり。 アドレス　対応する拡張ROMのアドレス 04D0　D0000～D0FFFH 04D1　D1000～D1FFFH 04D2　D2000～D2FFFH 04D3　D3000～D3FFFH 04D4　D4000～D4FFFH 04D5　D5000～D5FFFH 04D6　D6000～D6FFFH 04D7　D7000～D7FFFH 04D8　D8000～D8FFFH (初代・E・F・Mは未使用) 04D9　D9000～D9FFFH (初代・E・F・Mは未使用) 04DA　DA000～DAFFFH (初代・E・F・Mは未使用) 04DB　DB000～DBFFFH (初代・E・F・Mは未使用) 04DC　DC000～DCFFFH (初代・E・F・Mは未使用) 04DD　DD000～DDFFFH (初代・E・F・Mは未使用) 04DE　DE000～DEFFFH (初代・E・F・Mは未使用) 04DF　DF000～DFFFFH (初代・E・F・Mは未使用) 04E0～04E5 未使用 04E6　E6000～E6FFFH 04E7　E7000～E7FFFH 04E8　E8000～E8FFFH 04E9　E9000～E9FFFH 04EA　EA000～EAFFFH 04EB　EB000～EBFFFH 04EC　EC000～ECFFFH 04ED　ED000～EDFFFH 04EE　EE000～EEFFFH 04EF　EF000～EFFFFH 04F0　F0000～F0FFFH(XAのみ) 04F1　F1000～F1FFFH(XAのみ) 04F2　F2000～F2FFFH(XAのみ) 04F3　F3000～F3FFFH(XAのみ) 初代・E・F・MではD0000～D7FFFH、それ以外のノーマルモードではD0000～DFFFFHの範囲で4Kバイトごとに拡張BIOSの存在をチェックする。 ハイレゾモードではE6000～EFFFFHの範囲で4Kバイトごとに拡張BIOSの存在をチェックする。XAだけはF0000～F3FFFHも拡張ROM領域として使用することができる。
04D0～ 04EFH	[LT・HA] ‡ 不明。XROM IDテーブルとは別用途
04F0H	カーソル描画制御フラグ [LT] † bit3　VSYNC時にカーソル状態の変更を抑制するフラグ bit2　カーソルの反転状態
04F2～ 04F3H	カーソル位置 [LT] † 仮想テキストVRAM上のオフセットアドレス

アドレス	意味
04F8〜 04FDH	ジャンプコード用ワークエリア（ITF、SCSI BIOS 等で使用）‡
0500H	BIOS 制御用フラグ（0）（BIOS_FLAG）
0501H	BIOS 制御用フラグ（1）（BIOS_FLAG）

0500H BIOS 制御用フラグ（0）（BIOS_FLAG）

ビット	意味
bit7	スタートアップタイプ （1→ウォームスタート／0→コールドスタート）STOP キーを押しながらリセットして起動すると 1
bit6	BASIC タイプ認識 （1→DISK BASIC、MS-DOS 版 BASIC 実行中とその子プロセス／0→ROM BASIC、MS-DOS、その他の OS）
bit5	キーバッファオーバーフロー通知制御 1→オーバーフロー時にビープを鳴らさない 0→オーバーフロー時にビープを鳴らす
bit4	640K バイト以内での増設 RAM の有無［ノーマル］（1→あり／0→なし）
bit3	未使用† 1 なら VSYNC 検出中とされているが、実際に BIOS ROM を調べてみてもこのビットを操作、参照している部分は見つからない。053CH bit6 が相当する動作をする。
bit2	640K バイトフロッピーディスクインターフェイスのモード （1→AI Enable／0→AI Disable）
bit1	両用フロッピーディスクインターフェイスモード［ノーマル］ 1→1M バイトインターフェイスモード 0→640K バイトインターフェイスモード
bit0	機種情報（1）（初代・E・F・M・LT・HA 判別） （1→その他／0→初代・E・F・M・LT・HA・ハイレゾモード）

0501H BIOS 制御用フラグ（1）（BIOS_FLAG）

ビット	意味
bit7	システムクロック 1→8MHz 系（タイマクロック 2MHz） 0→10MHz 系（タイマクロック 2.5MHz）
bit6	CPU タイプ （1→V30・V30HL・V50・V33・V30 モード／0→80x86）
bit5	機種情報（2）（初代・XA 判別） （1→その他／0→初代・XA）
bit4	機種情報（3）（U・LT・HA 判別） （1→U・LT・HA／0→その他）
bit3	機種情報（4）（ハイレゾ・ノーマル判別） 1→ハイレゾモード 0→ノーマルモード
bit2〜0	リアルモードメモリサイズ

値	意味
101B	768K バイト
100B	640K バイト
011B	512K バイト
010B	384K バイト
001B	256K バイト
000B	128K バイト

アドレス	意味
0502H	キーボードバッファ（KB_BUF） 入力されたキーコードを 2 バイト単位で最大 16 個格納する。ハイレゾでは起動時のキーバッファ。
0522H	キーコード変換テーブルのオフセット（KB_SHIFT_TBL）［ノーマル］ 現在のシフト状態のキーコード変換テーブルのオフセット SHIFT、CTRL、CAPS、GRPH、カナが ON/OFF されるごとに書き換えられる。キーコードを JIS8 単位コードに変換するのに用いる。セグメントは KB_CODE（05C6H）と同じ。
0522H	キーバッファサイズ（KB_BUFFER_SIZ）［ハイレゾ］ バッファに格納できるキー情報（2 バイト一組）の個数。起動時は 10H。
0523H	キーボードフラグ（KB_FLAG）［ハイレゾ］ XA・RL の BIOS 内では使用されていない。
0524H	キーバッファのリードポインタ（KB_BUF_HEAD） キーコードバッファの格納済みエリアの先頭オフセットアドレス。ハイレゾのとき、セグメントは KB_BUF_ADR（0410H）と同じ。
0526H	キーバッファのライトポインタ（KB_BUF_TAIL） キーコードバッファ格納済みエリアの最終オフセットアドレス＋1。ハイレゾのとき、セグメントは KB_BUF_ADR（0410H）と同じ。
0528H	キーバッファ内のキーコード数（KB_COUNT） キーコードバッファ内に格納済みのキーコードの数
0529H	キーボードのエラーリトライ回数（KB_RETRY） キーボードにデータ再送信を要求した回数
052A〜 0539H	キーの押下状態テーブル（KB_KY_STS） キーが押されるとつぎの表に示したように各キーに対応したビットが 1 になる

052AH	キー	052BH	キー	052CH	キー	052DH	キー
bit7	'7	bit7	TAB	bit7	I	bit7	D
bit6	&6	bit6	BS	bit6	U	bit6	S
bit5	%5	bit5	¦\	bit5	Y	bit5	A
bit4	$4	bit4	‘^	bit4	T	bit4	RET
bit3	#3	bit3	=-	bit3	R	bit3	{[
bit2	"2	bit2	0	bit2	E	bit2	~@
bit1	!1	bit1	)9	bit1	W	bit1	P
bit0	ESC	bit0	(8	bit0	Q	bit0	O

052EH	キー	052FH	キー	0530H	キー	0531H	キー
bit7	*:	bit7	M	bit7	R.DN	bit7	HELP
bit6	+;	bit6	N	bit6	R.UP	bit6	H/CLR
bit5	L	bit5	B	bit5	XFER	bit5	↓
bit4	K	bit4	V	bit4	SPC	bit4	→
bit3	J	bit3	C	bit3	_	bit3	←
bit2	H	bit2	X	bit2	?/	bit2	↑
bit1	G	bit1	Z	bit1	>.	bit1	DEL
bit0	F	bit0	}]	bit0	<,	bit0	INS

0532H	キー	0533H	キー	0534H	キー	0535H	キー
bit7	5	bit7	,	bit7		bit7	
bit6	4	bit6	0	bit6	vf・5	bit6	HOME
bit5	*	bit5	=	bit5	vf・4	bit5	
bit4	9	bit4	3	bit4	vf・3	bit4	
bit3	8	bit3	2	bit3	vf・2	bit3	
bit2	7	bit2	1	bit2	vf・1	bit2	
bit1	/	bit1	/	bit1	NFER	bit1	
bit0	–	bit0	6	bit0		bit0	

0536H	キー	0537H	キー	0538H	キー	0539H	キー
bit7	f・6	bit7		bit7		bit7	
bit6	f・5	bit6		bit6		bit6	
bit5	f・4	bit5		bit5		bit5	
bit4	f・3	bit4		bit4	CTRL	bit4	
bit3	f・2	bit3	f・10	bit3	GRPH	bit3	
bit2	f・1	bit2	f・9	bit2	カナ	bit2	
bit1	COPY	bit1	f・8	bit1	CAPS	bit1	
bit0	STOP	bit0	f・7	bit0	SHIFT	bit0	

053AH シフトキー押下状態フラグ（KB_SHFT_STS）

ビット	意味
bit4	CTRL
bit3	GRPH
bit2	カナ
bit1	CAPS
bit0	SHIFT

（1→押されている／0→押されていない）
ハイレゾの場合、0408〜040FH のシフトキーコードテーブルに登録されたキーが押されたとき、対応するビットが 1 になる。

アドレス	意味
053BH	CRT の行当りラスタ本数－1（CRT_RASTER）［ノーマル・ハイレゾ］

アドレス	意味
053CH	テキスト画面ステータスフラグ（CRT_STS_FLAG）［ノーマル］

ビット	意味
bit7	CRT タイプ 1→高解像 CRT（88 タイプ） 0→標準 CRT（80 タイプ）
bit6	VSYNC 検出 BIOS 内の VSYNC 待ちルーチンで使用する。VSYNC 待ちルーチンに入ると 1 にセットされ、ここが 0 になるまでループする。VSYNC 割り込みが発生すると割り込み処理ルーチンが 0 クリアする。
bit5	
bit4	
bit3	KCG（漢字キャラクタジェネレータ）アクセスモード （1→ドットアクセス／0→コードアクセス）
bit2	アトリビュートモード 1→簡易グラフィック 0→バーティカルライン
bit1	表示桁数 （1→40 カラム／0→80 カラム）
bit0	表示行数 （1→20 行／0→25 行）

アドレス	意味
053DH	作業用カウンタ（CRT_CNT）［ノーマル・ハイレゾ］ CRT BIOS で使用
053E〜0541H	CRT 制御用パラメータブロックのアドレス（CRT_OFST/CRT_SEG_ADR）［ノーマル・ハイレゾ］
0542〜0545H	VSYNC 割り込み（INT 0AH）ベクタの待避領域（CRT_V_INT_OFST/SEG）［ノーマル・ハイレゾ］ BIOS 内の VSYNC 待ちルーチンで VSYNC 割り込みをトラップするとき、元の割り込み先をここに保存する。
0542H	グラフィック VRAM 上での一行のバイト数［LT］† （取りうる値は 80 × 16 または 80 × 20）
0546H	CG から読み出す文字フォントパターン識別（CRT_FONT）［ノーマル］

ビット	意味
bit1	タイプ（1→漢字／0→ANK 文字）
bit0	フォントサイズ（1→7 × 11／0→6 × 7）

アドレス	意味
0546H	テキスト画面ステータスフラグ（CRT_FLAG）［ハイレゾ］

ビット	意味
bit7	VSYNC 検出 BIOS 内の VSYNC 待ちルーチンで使用する。VSYNC 待ちルーチンに入ると 0 にセットされ、ここが 1 になるまでループする。VSYNC 割り込みが発生すると割り込み処理ルーチンが 1 にする。
bit4	表示行数（1→31 行／0→25 行）
bit3	KCG（漢字キャラクタジェネレータ）アクセスモード （1→ドットアクセス／0→コードアクセス）
bit0	フォントタイプ（1→全角タイプ／0→半角タイプ（ANK 含む））

アドレス	意味
0547H	テキスト画面の分割数（CRT_WINDOW_NO）［ノーマル・ハイレゾ］ GDC に設定する部分画面（スクロールエリア）の個数
0548〜0549H	テキスト VRAM の表示開始アドレス（CRT_W_VRAMADR）［ノーマル・ハイレゾ］ CPU から見えるテキスト VRAM のオフセットアドレスが格納される
054A〜054BH	表示画面全体のラスタ数（CRT_W_RASTER）［ノーマル・ハイレゾ］ 200 ラインか 400 ラインかを示す。ただし、正しく設定されていない場合がある模様。
054CH	グラフィック画面ステータスフラグ（PRXCRT）［ノーマル］

ビット	意味
bit7	CRT 状態（1→表示／0→表示停止）
bit6	CRT タイプ 1→高解像 CRT（88 タイプ） 0→標準 CRT（80 タイプ）
bit5	H98 CRT タイプ† 1→H98 専用（N5926-01、02） 0→従来型
bit4	H98 CRT モード† （1→拡張／0→標準）
bit3	ユーザ定義文字数 （1→188 文字／0→63 文字）
bit2	拡張グラフィック VRAM（E0000〜E7FFFH のグラフィック VRAM） （1→あり／0→なし）
bit1	GRCG（1→あり／0→なし）
bit0	DIP SW 1-8（1→ON／0→OFF） DIP SW の状態がコピーされるだけなので、このビットでグラフ LIO が 16 色モードをサポートしているかどうかの判定はできない。

アドレス	意味
054DH	グラフィックモード（PRXDUPD）†［ノーマル・ハイレゾ］

ビット	意味
bit7	H98 256 色ボードの有無 （1→あり／0→なし）
bit6	EGC 使用可否（1→可／0→不可） NS/E ではレジューム機能を使うと EGC が使えなくなる
bit5	DIP SW 2-8（GDC CLOCK）［ノーマル］ 1→ON（5MHz） 0→OFF（2.5MHz）
bit4	グラフィック VRAM の種類［ノーマル］ 1→デュアルポート RAM 0→非デュアルポート RAM
bit3	高速書込みモード［ノーマル］ （1→ON／0→OFF） 1 のとき、グラフィック VRAM が非デュアルポート RAM の機種では GDC で描画するとちらつきが出る
bit2	GDC クロック［ノーマル］ （1→5MHz／0→2.5MHz） GDC の実際のクロック。GDC は 200 ラインモードのとき必ず 2.5MHz になる。
bit1〜0	GDC のドット修正モードの設定状態

値	意味
11B	SET
10B	CLEAR
01B	COMPLEMENT
00B	REPLACE

アドレス	意味
054EH	GDC に設定した線種パターン情報（PRXGLS） GDC の TEXTW コマンドで GDC 内の RAM に格納する線種パターン
054EH	GDC に設定したグラフィック文字パターン情報（PRXGCPTN） GDC の TEXTW コマンドで GDC 内の RAM に格納するグラフィック文字パターン
0556〜0559H	RS-232C BIOS CH0 受信バッファのアドレス（RS_CH0_OFST/SEG）
055AH	ODA 系プリンタのシフト状態（PR_SIFT）

ビット	意味
bit2	ODA プリンタからの割り込み （1→あり／0→なし）
bit1	シフト状態 （1→状態確定／0→状態不確定）
bit0	シフト状態（1→SO 状態／0→SI 状態）

アドレス　意　味

055BH
RS-232C BIOS 受信データのシフト状態 (RS_S_FLAG)

ビット	意　味
	SI/SO 変換（1→あり／0→なし（既定値））
bit7	CH 0
bit6	CH 1
bit5	CH 2
	SI/SO シフト状態（1→SO／0→SI（既定値））
bit4	CH 0
bit3	CH 1
bit2	CH 2

055C〜055DH
フロッピーディスク、SASI ハードディスク接続状態 (DISK_EQUIP)

ビット	意　味
bit15	640K バイトフロッピーディスクユニット#3
bit14	640K バイトフロッピーディスクユニット#2
bit13	640K バイトフロッピーディスクユニット#1
bit12	640K バイトフロッピーディスクユニット#0
bit11	SASI ハードディスクユニット#3 [H98]
bit10	SASI ハードディスクユニット#2 [H98]
bit9	SASI ハードディスクユニット#1
bit8	SASI ハードディスクユニット#0
bit7	320K バイトフロッピーディスクユニット#3
bit6	320K バイトフロッピーディスクユニット#2
bit5	320K バイトフロッピーディスクユニット#1
bit4	320K バイトフロッピーディスクユニット#0
bit3	1M バイトフロッピーディスクユニット#3
bit2	1M バイトフロッピーディスクユニット#2
bit1	1M バイトフロッピーディスクユニット#1
bit0	1M バイトフロッピーディスクユニット#0

（1→接続／0→未接続）
DISK BIOS の INITIALIZE コマンド実行時に、接続されているディスク装置の状態がセットされる。
1M バイト/640K バイトフロッピーディスク両用インターフェイスの場合、設定されているインターフェイスモードのビットのみがセットされる。

055E〜055FH
ディスク装置からの割り込み状況を示すフラグ (DISK_INT)

ビット	意　味
bit15	640K バイトフロッピーディスクユニット#3
bit14	640K バイトフロッピーディスクユニット#2
bit13	640K バイトフロッピーディスクユニット#1
bit12	640K バイトフロッピーディスクユニット#0
bit8	SASI ハードディスクユニット
bit3	1M バイトフロッピーディスクユニット#3
bit2	1M バイトフロッピーディスクユニット#2
bit1	1M バイトフロッピーディスクユニット#1
bit0	1M バイトフロッピーディスクユニット#0

（1→対応するデバイスから割り込みが発生／0→割り込みなし）

0560H
320K バイトフロッピーディスクタイプ (DISK_TYPE)

値	ディスクタイプ
FFH	片面タイプ
EFH	両面タイプ
00H	なし

0561H
320K バイトフロッピーディスク動作モード (DISK_MODE)

ビット	意　味
bit3	ユニット#3
bit2	ユニット#2
bit1	ユニット#1
bit0	ユニット#0

（1→両面／0→片面）

0562H
320K バイトフロッピーディスクタイムアウト時間 (DISK_TIME)
320K バイトフロッピーディスクからのブート時、タイムアウト処理を行う。単位はミリ秒。0 のときは完了するまで無条件に待つ。POWER ON/RESET 時、約 35 秒に設定。

アドレス　意　味

0564〜0583H
1M バイトフロッピーディスクインターフェイスの FDC のリザルトステータス (DISK_RESULT)
1M バイトフロッピーディスクドライブの I/O 終了時、FDC から返されるステータス情報 1 ユニットに対し 8 バイト× 4 ユニット分ある。SEEK 系コマンドでは ST0 と PCN のみ通知される。

	7	6	5	4	3	2	1	0	
+0	IC1	IC0	SE	EC	NR	HD	US1	US0	*1
+1	EN	0	DE	OR	0	ND	NW	MA	*2
+2	0	CM	DD	NC	SH	SN	BC	MD	*3
+3	0〜76								*4
+4	0〜1								*5
+5	1〜26								*6
+6	0〜3								*7
+7	0〜76								*8

*1 ST0リザルトステータス
ST0のbit5(SE)=1のときPCNとして使用する
*2 ST1リザルトステータスまたはPCNプレゼントシリンダナンバ
*3 ST2リザルトステータス
*4 Cシリンダ番号
*5 Hヘッド番号
*6 Rレコード番号
*7 Nデータ長
*8 NCN現在のシリンダ番号

0564〜056BH
1M バイトフロッピーディスクユニット#0

056C〜0573H
1M バイトフロッピーディスクユニット#1

0574〜057BH
1M バイトフロッピーディスクユニット#2

057C〜0583H
1M バイトフロッピーディスクユニット#3

0584H
OS がブートしたディスク装置の DA/UA (DISK_BOOT)
両用フロッピーディスクからブートしたときは、ブートディスクのメディアを示す値になる。(例：640K バイトインターフェイスモードで 1M バイトフロッピーディスクから起動したら F0H)

0585H
SASI ハードディスクから戻される完了時ステータス情報 (DISK_STATUS)

ビット	意　味
bit7	LVN
bit6	LVN
bit5	LVN
bit1	エラー
bit0	パリティエラー

0586〜0589H
SASI ハードディスクから返されるセンス情報 (DISK_SENSE)
固定ディスクの Request Sense コマンドで通知される Sense Status Bytes を格納する。

アドレス	意　味
0586H	センスバイト
0587H	論理アドレス H（上位 2 ビットは 0）
0588H	論理アドレス M
0589H	論理アドレス L

058AH
タイマ BIOS のインターバルタイマカウンタ (CA_TIM_CNT)
ノーマルモードとハイレゾのシングルイベントタイマで使用。

058CH
(DISK_WORK)

058EH
PAINT 処理の高速制御用エリア (PRXGDCQ)
グラフィック LIO、グラフィック BIOS で使用

0590H
H98 DISK BIOS モード

ビット	意　味
bit7	SASI BIOS アクセスモード 1→インダイレクトモードアクセス 0→ダイレクトモードアクセス

アドレス　意　味

bit6　SASI BIOS インダイレクトモードサポート（1→あり／0→なし）

bit5　フロッピーディスク BIOS アクセスモード
1→インダイレクトモードアクセス
0→ダイレクトモードアクセス

bit4　フロッピーディスク BIOS インダイレクトモードサポート（1→あり／0→なし）

bit3　SCSI BIOS アクセスモード
1→インダイレクトモードアクセス
0→ダイレクトモードアクセス

bit2　SCSI BIOS インダイレクトモードサポート（1→あり／0→なし）

0596H　H98SCPU 速度 ‡

値	対応するスピード
00H	80486SX 20MHz
01H	80386 20MHz 相当
09H	80386 20MHz 相当

0598H　DF → NUM キー無効
DE → NUM キー有効

059AH　98NOTE のハードディスク関係の情報が格納される ‡

059CH　NOTE RAM バンク用 0E8EH ポート保存［NS/E］@
NMI 処理中に NOTE RAM のバンクを切り替えるとき、0E8EH ポートの内容を保存する。

059CH　JEIDA メモリカードのサイズ（CARD_SIZE）［HA］

ビット	意　味
bit15～8	リザーブ
bit7～3	仮数部（Y） Y＝1FH →仮数は 32 Y＝00H →仮数は 1
bit2～0	指数部（X）

X	単位	最大容量（バイト）
7	拡張	無効
6	2M	64M
5	512K	16M
4	128K	4M
3	32K	1M
2	8K	256K
1	2K	64K
0	512	16K

（容量＝$(Y+1) \times 0.5 \times 4^{X}$）

059DH　NOTE RAM バンク用 1E8EH ポート保存［NS/E］@
NMI 処理中に NOTE RAM のバンクを切り替えるとき、1E8EH ポートの内容を保存する。

059EH　NOTE 特殊機能の起動指示フラグ［NS/E］@

ビット	意　味
bit7	（1→電源 OFF ／ 0→無動作）
bit3	（1→ノートメニューを起動／ 0→無動作）

NMI 処理中にこのワークを参照して実行する。
このビットを立てるのは INT 0EH（100 ミリ秒周期タイマ）処理ルーチン。
その他のビットも機能を割り付けてあるが不明。

059EH　JEIDA メモリカードの状態（CARD_STATUS）［HA］

ビット	意　味
bit7	アクセス状態 1→アクセス中 0→アクセスしていない
bit6	リード／ライト（bit7 ＝ 1 のとき有効） （1→ライト／ 0→リード）
bit5	カード有効／無効（bit0～1 ＝ 00B のとき認識前） （1→カード無効／ 0→カード有効）
bit4	カード入れ換え検出 （1→入れ換えた／ 0→入れ換えない）
bit3	カード有無 （1→あり／ 0→なし）
bit2	ライトプロテクト（EMS では常に 1） （1→あり／ 0→なし）
bit1	RAM/ROM カード （1→あり／ 0→なし）
bit0	EMS カード （1→あり／ 0→なし）

アドレス　意　味

05A3　HA の INT　10～17H ルーチン内で OUT　8810H、［0000:05A3］が行われている。‡

05C0H　DIP SW 2 の設定状態のコピー（BASIC_DIPSW）
ただし、この値は BASIC インタプリタ（ROM 版、DISK 版、MS-DOS 版）が設定するので MS-DOS 起動時は 0。

05C1H　RS-232C BIOS DEL コード制御（RS_D_FLAG）
RS-232C DEL 受信時の処理
1 →メモリスイッチ 3 bit7 ＝ 1 の設定に従う
0 → DEL コードをそのままバッファに格納

ビット	意　味
bit2	CH 2
bit1	CH 1
bit0	CH 0

05C2H　GP-IB の制御情報通知領域へのポインタ（GPIBWORK）

05C6H　キーボードのコード変換テーブルへのポインタ（KB_CODE）［ノーマル］

05CAH　640K バイトフロッピーディスクドライブの接続されている各ユニットのアクセスモード（F2DD_MODE）

ビット	意　味
bit7	ユニット #3
bit6	ユニット #2
bit5	ユニット #1
bit4	ユニット #0

（1→倍密（96tpi）／ 0→単密（48tpi））

bit3	ユニット #3
bit2	ユニット #2
bit1	ユニット #1
bit0	ユニット #0

（1→両面／ 0→片面）
初期値 ＝ FFH（2DD/両面）

05CBH　640K バイト、両用フロッピーディスクドライブのモータオフまでの時間（単位 100 ミリ秒）（F2DD_COUNT）
フロッピーディスクインターフェイス初期化コマンド実行後は 150（15 秒）に設定される。05D7H にコピーしてカウントダウンする。両用フロッピーディスクドライブの 1M バイトインターフェイスモードでは、モータオフ制御機能を持ったモデルでのみ有効。

05CC～05CFH　640K バイトフロッピーディスクパラメータテーブルポインタへのポインタ（F2DD_POINTER）
640K バイトフロッピーディスクドライブのパラメータテーブルへのポインタテーブルの位置を示すポインタ各ユニット毎にオフセットアドレスのポインタが用意されているが、これらは全て同一のテーブルを指している。

F2DD_POINTER

OFS	SEG

ポインタテーブル

FD#0 OFS	FD#1 SEG	FD#2 OFS	FD#3 SEG

パラメータテーブル

	MFM				FM			
	R/W		Format		R/W		Format	
	EOT	GLP	SC	GPL	EOT	GLP	SC	GPL
+00H	00H	00H	00H	00H	1AH	07H	1AH	1BH
+08H	1AH	0EH	1AH	36H	0FH	0EH	0FH	2AH
+10H	0FH	1BH	0FH	54H	08H	1BH	08H	3AH
+18H	08H	35H	08H	74H	00H	00H	00H	00H

アドレス	意 味

05D0〜
05DFH　640KバイトフロッピーディスクインターフェイスのFDCリザルトステータス（F2DD_RESULT）
640Kバイトフロッピーディスクドライブの I/O 終了時、FDCから返されるステータス情報

05D0〜05D6H　READ/WRITE系コマンド

アドレス	意 味
05D0H	ST0 リザルトステータス
05D1H	ST1 リザルトステータス
05D2H	ST2 リザルトステータス
05D3H	C シリンダ番号
05D4H	H ヘッド番号
05D5H	R レコード番号
05D6H	N データ長

05D7H　モータオフカウンタ
タイマ割り込み毎にデクリメントし0になったらモータオフ
（詳細は05CBHを参照）

05D8〜05DFH　SEEK/RECALIBRATEコマンド

アドレス	意 味
05D8H	ST0 ユニット#0 リザルトステータス
05D9H	PCN ユニット#0 プレゼントシリンダナンバ
05DAH	ST0 ユニット#1 リザルトステータス
05DBH	PCN ユニット#1 プレゼントシリンダナンバ
05DCH	ST0 ユニット#2 リザルトステータス
05DDH	PCN ユニット#2 プレゼントシリンダナンバ
05DEH	ST0 ユニット#3 リザルトステータス
05DFH	PCN ユニット#3 プレゼントシリンダナンバ

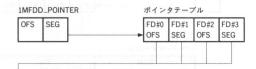
05E0H　サウンドBIOS用ワークエリア（MUSIC_WORK）

05E8〜
05EBH　SASI#0 パラメータテーブルへのポインタ‡
新センス（AH = 84H、INT 1BH）サポートのSASI BIOSのパラメータポインタ

05EC〜
05EFH　SASI#1 パラメータテーブルへのポインタ‡
新センス（AH = 84H、INT 1BH）サポートのSASI BIOSのパラメータポインタ

05F0〜
05F3H　RS-232C CH1受信バッファポインタ（RS_CH1_OFST/SEG）

05F4〜
05F7H　RS-232C CH2受信バッファポインタ（RS_CH2_OFST/SEG）

05F8〜
05FBH　1Mバイトフロッピーディスクのパラメータテーブルへのポインタテーブルの位置を示すポインタ

パラメータテーブル

	FM				MFM			
	R/W		Format		R/W		Format	
	EOT	GLP	SC	GPL	EOT	GLP	SC	GPL
+00H	00H	00H	00H	00H	1AH	07H	1AH	1BH
+08H	1AH	0EH	1AH	36H	0FH	0EH	0FH	2AH
+10H	0FH	1BH	0FH	54H	08H	1BH	08H	3AH
+18H	08H	35H	08H	74H	00H	00H	00H	00H

パラメータテーブルのセグメントはポインタテーブルと同じ。
各ユニット毎にオフセットアドレスのポインタが用意されているが、実際には全て同一のテーブルを指している。

アドレス	意 味

05FCH　マウスBIOS作業域のセグメントアドレス（MOUSEW）［ハイレゾ］

05FEH　BASIC LIOのDATAセグメントアドレス（BASIC_LDSR）
（N88-BASIC動作時）
MS-DOS版N88-BASICの子プロセスではこの番地が0以外の値を持っており、再度のN88BASICの起動ができなくなる。

MS-DOSのワークエリア 0060:0000H〜

● MS-DOSのIO.SYSのワークエリアを参照すると、MS-DOSの現在の状態が取得できる。

アドレス	意 味

0060H　MS-DOS内部ワークエリア一覧
このセグメントの内容は日電から一切公開されていない
一部のワークエリアはMS-DOSの製品バージョンによって異なる。

0030H　NEC 0BバンクEMSの自動切り換えフラグ
00H→プロセス起動時、0Bバンクを自動的にグラフィックVRAMに切り換える
01H→プロセス起動時、0Bバンクを自動的にグラフィックVRAMに切り換えない

0031H　実装プロテクトメモリ容量
使用／未使用に関らず、実装しているプロテクトメモリの容量が格納されている。

0052〜
0053H　ADDDRVの常駐セグメント

0068H　RSDRV.SYSの使用するAUXプロトコル（1）

ビット	意 味
bit7〜6	ストップビット長（01B→1ビット／11B→2ビット）
bit5	パリティ（1→偶数／0→奇数）
bit4	パリティ有無（1→あり／0→なし）
bit3〜2	データビット長（10B→7ビット／11B→8ビット）
bit1	通信方式（0→全二重／1→半二重）
bit0	Xフロー制御の有無（1→あり／0→なし）

起動時、メモリスイッチ1の状態がコピーされる。
INT DCによる設定がここに反映される。

0069H　RSDRV.SYSの使用するAUXプロトコル（2）
bit3〜0　通信速度
0001B → 75bps
0010B → 150bps
0011B → 300bps
0100B → 600bps
0101B → 1200bps
0110B → 2400bps
0111B → 4800bps
1000B → 9600bps
起動時、メモリスイッチ2の状態がコピーされる。
INT DCによる設定がここに反映される。

006C〜
007BH　A:〜P:のDA/UA
CL = 13H、INT DCHで得られるリストの0000〜000Fの内容

008AH　漢字／グラフモードフラグ
00H→グラフ文字モード
（エスケープシーケンス'^［）3'発行時）
01H→漢字モード
（エスケープシーケンス'^［）0'発行時）

008BH　グラフ文字モード表示マーク
グラフ文字モードのとき、画面左下に表示される文字。
通常は'g'（67H）が格納される。書き換えると画面クリア時に表示が変化する。
漢字モードのときは、' '（20H）が格納されている。

アドレス	意　味
008CH	シフトファンクション表示マーク シフトファンクション表示状態のとき、画面左下に表示される文字。 通常は'*'（2AH）が格納される。書き換えると画面クリア時に表示が変化する。 シフトファンクション表示状態でないときは、' '（20H）が格納されている。
00A4H	STOP キー割り込み再入防止フラグ
0110H	カーソル位置 Y 座標 画面の上端を 0 とした行位置（PS98–121、PS98–XA125 では異なる）
0111H	ファンクション表示状態フラグ（PS98–125～） 00H →ファンクションキー無表示状態 01H →ファンクションキー表示状態 02H →シフトファンクションキー表示状態
0112H	スクロール範囲下限 画面の上端を 0 とした行位置 スクロール範囲下限の直下の行にファンクションが表示される。 通常はファンクションキー表示部を除いた画面表示行数–1 と等しい。
0113H	画面ライン数 00H →ノーマル 20 行／ハイレゾ 25 行 01H →ノーマル 25 行／ハイレゾ 31 行 この値を書き換えてライン数を変化させることはできない
0114H	消去アトリビュート テキスト VRAM のアトリビュート
0115H	漢字上位バイト格納フラグ 00H →通常状態 01H →漢字の下位バイト待ち状態
0116H	漢字上位バイト格納用ワーク 漢字の 1 バイト目が来たらここに上位バイトを格納し、0115 にフラグを立て下位バイトを待つ。
0117H	行折り返しフラグ 00H → 1 行が 80 桁を超えたとき、カーソルを 1 行下の左端に移動 エスケープシーケンスの'^ [[?7H' で設定 01H → 1 行が 80 桁を超えたとき、超えた部分を表示しない エスケープシーケンスの'^ [[?7l' で設定
0118H	スクロールスピード 00H →ノーマル 01H →スロー
0119H	消去キャラクタ 初期状態は 20H。
011BH	カーソル表示状態フラグ 00H →表示なし 01H →表示あり 書き換えただけでは変わらないが、画面クリアで反映する。
011CH	カーソル位置の X 座標 画面の左端を 0 とした桁位置（PS98–121、PS98–XA125 では異なる）
011DH	表示アトリビュート テキスト VRAM のアトリビュート
011EH	スクロール範囲上限 画面の上端を 0 とした行位置
0126H	セーブカーソル Y 座標 エスケープシーケンスの'^ [[s' でセーブしたカーソル状態 画面の上端を 0 とした行位置
0127H	セーブカーソル X 座標 エスケープシーケンスの'^ [[s' でセーブしたカーソル状態 画面の左端を 0 とした桁位置
0128H	エスケープシーケンスデータカウンタ 受け取ったデータがエスケープシーケンスのとき、キャラクタのカウンタとして使われる。 受け取ったエスケープシーケンスは、0134 に格納されているアドレスに保存される。
012BH	セーブカーソルアトリビュート エスケープシーケンスの'^ [[s' でセーブしたカーソル状態 テキスト VRAM のアトリビュート
0134～ 0135H	エスケープシーケンスバッファへのポインタ ここで示されるアドレスに、エスケープシーケンスのデータが一時的に格納されている。
013C～ 013DH	拡張アトリビュート対応表示アトリビュート テキスト VRAM のアトリビュート
013E～ 013FH	拡張アトリビュート対応消去アトリビュート テキスト VRAM のアトリビュート
018DH	カーソル位置（Y） 画面の上端を 0 とした行位置（PS98–XA125 の場合）
018EH	カーソル位置（X） 画面の左端を 0 とした桁位置（PS98–XA125 の場合）
2820～ 2823H	A:～Z:までの DA/UA リストへのポインタ（PS98–011～） CL = 13H、INT DCH で得られるリストの 001A～004D の内容

メモリスイッチ

● メモリスイッチは、システムの状態を保持する不揮発性のメモリである。

	ノーマル	ハイレゾ	LT・HA
MSW0			D000:3800H
MSW1	A000:3FE2H	E000:3FE2H	D000:3801H
MSW2	A000:3FE6H	E000:3FE6H	D000:3802H
MSW3	A000:3FEAH	E000:3FEAH	D000:3803H
MSW4	A000:3FEEH	E000:3FEEH	D000:3804H
MSW5	A000:3FF2H	E000:3FF2H	D000:3805H
MSW6	A000:3FF6H	E000:3FF6H	D000:3806H
MSW7	A000:3FFAH	E000:3FFAH	D000:3807H
MSW8	A000:3FFEH	E000:3FFEH	D000:3808H
MSW9			D000:3809H
:			:
MSW255			D000:38FFH

■ メモリスイッチ 1

ビット	意　味
bit7～6	ストップビット長 （01B → 1 ビット／ 10B → 1.5 ビット／ 11B → 2 ビット）
bit5	パリティ（1 →偶数／ 0 →奇数）
bit4	パリティ有無（1 →あり／ 0 →なし）
bit3～2	データビット長（10B → 7 ビット／ 11B → 8 ビット）
bit1	通信方式（0 →全二重／ 1 →半二重）
bit0	X フロー制御の有無（1 →あり／ 0 →なし）

MS–DOS の SPEED コマンド、BASIC、ターミナルモードの通信パラメータの初期値を設定する。
bit1 はターミナルモードでのみ使用。

■ メモリスイッチ 2

ビット	意味
bit7	Sパラメータの有無（1→あり／0→なし）
bit6	リターンキーで送出するコード （1→CR+LF／0→CR）
bit5	CR受信時の処理（1→CR／0→CR+LF）
bit4	日本語シフトコード（1→（KI＝1A70H、KO＝1A71H）／0→（KI＝1B4BH、KO＝1B48H））
bit3～0	通信速度 0001B→75bps 0010B→150bps 0011B→300bps 0100B→600bps 0101B→1200bps 0110B→2400bps 0111B→4800bps 1000B→9600bps

MS-DOSのSPEEDコマンド、BASIC、ターミナルモードの通信パラメータの初期値を設定する。
bit7はBASICとターミナルモードのみで使用。
bit6～bit4はターミナルモードのみで使用。

■ メモリスイッチ 3

ビット	意味
bit7	DEL受信時の処理（BASIC：1→NUL／0→DEL） （ターミナルモード：1→NUL／0→BS）
bit6	テキスト画面の色指定（1→緑／0→白）
bit5	V30用コプロセッサのクロック （1→8MHz/16MHz／0→10MHz）［ノーマル］ 新キーボード、CAPSロック状態 （1→ロック／0→アンロック）［ハイレゾ］
bit4	V30用コプロセッサの有無 （1→あり／0→なし）［ノーマル］
bit3	8086、80286、80386用コプロセッサの有無 （1→あり／0→なし）
bit2～0	メモリサイズ （000B→128Kバイト／001B→256Kバイト／ 010B→384Kバイト／011B→512Kバイト／ 100B→640Kバイト／101B→768Kバイト）

bit7はBASICとターミナルモードでのみ使用。
bit6←1にすると、テキスト画面の緑と白の色属性が入れ代わる。
bit5はテクニカルデータブックによるとMS-DOSでのみ機能するとされているが、その意味は不明。
bit4～3はコプロセッサ搭載の目印に過ぎず、この値によってハードウェアの状態が変化することはない。CPU命令だけでコプロセッサの有無を判別するプログラムであればこの値を参照する必要はない。BASICなどはこの値を参照してコプロセッサの有無を判別しているが、そのようなプログラミングは避けたほうが良い。
bit2～0は640Kバイト（768Kバイト）以内のメモリ容量の設定。プロテクトモード用メモリにはこの種の設定は必要ない。

■ メモリスイッチ 4

ビット	意味
bit7	BASIC拡張ROM、CE000エントリ （1→あり／0→なし）
bit6	BASIC拡張ROM、CA000エントリ （1→あり／0→なし）
bit5	BASIC拡張ROM、D4000エントリ （1→あり／0→なし）
bit4	BASIC拡張ROM、D0000エントリ （1→あり／0→なし）
bit3	BASIC拡張ROM、CC000エントリ （1→あり／0→なし）
bit2	BASIC拡張ROM、C8000エントリ （1→あり／0→なし）
bit1	BASIC拡張ROM、C4000エントリ （1→あり／0→なし）
bit0	BASIC拡張ROM、C0000エントリ （1→あり／0→なし）

拡張ROMでBASICのコマンドを拡張するための設定。ROM BASIC、DISK BASICでのみ使用。MS-DOS版BASICでは設定不要。ROMが存在しない領域に対応するビットを1にしてBASICを起動しようとすると、存在しないROM領域に実行が移って暴走する。MS-DOSなどを起動する場合には無関係である。

■ メモリスイッチ 5

ビット	意味
bit7～4	ブート装置の指定 ［ノーマル］ 0000B→標準 （FD→SASI→SCSI→光ディスクの順にサーチ） 0010B→640KB FD 0100B→1MB FD 0110B→光ディスク 1010B→SASI HD#1 1011B→SASI HD#2 1100B→SCSI HD その他→ROM BASIC ［LT・HA］ 0000B→標準 （FD→メモリカード［HA］→ROMの順にサーチ） 1101B→メモリカード［HA］ 1110B→ROM DISK［LT・HA］ その他→ブート不可 ［ハイレゾ］ 0000B→フロッピー起動 0100B→ハードディスク起動 bit3はDISK BASIC、MS-DOS版BASICでのみ使用。 bit2～1はDISK BASICでのみ使用。 bit0はBASIC、MS-DOSのPRINT.SYSで使用。
bit3	画面ハードコピーモード（1→カラー／0→モノクロ）
bit2	固定ディスクユーザ識別名 （1→使用しない／0→使用する）
bit1	ディスクデバイス名順序 （1→HDD→FDD／0→FDD→HDD）
bit0	プリンタ（1→24ドット／0→16ドット）

■ メモリスイッチ 6

［ノーマル］

ビット	意味
bit7	新キーボード、カナロック状態 （1→ロック／0→アンロック）
bit6	新キーボード、CAPSロック状態 （1→ロック／0→アンロック）
bit5	電話制御機能（1→あり／0→なし）
bit4	拡張画面ハードコピー機能（1→あり／0→なし）
bit3	モニタモード（1→拡張／0→ROM内機能のみ）
bit2～0	未使用

bit7～6はRA以降の新キーボードを持つ機種が電源OFFしてもキーロック状態を保存するために使用する。
bit5、3はDISK BASICでのみ使用。
bit4はDISK BASICとMS-DOS版BASICでのみ使用。なお、DISK BASICとMS-DOS版BASICではこのビットの意味が反転するので注意。

[ハイレゾ]

ビット	意　味
bit7～4	起動領域のセレクタ番号 0000B→ESC（セレクタ0）／0001B→f・1（セレクタ1）／…／1111B→vf・5（セレクタ15）

■ メモリスイッチ7

[ハイレゾ]

ビット	意　味
bit7	新キーボード、カナロック状態 （1→ロック／0→アンロック）

■ メモリスイッチ8

μPD1990使用システムで「年」を保持。パックトBCD形式。（1992年ならば92H）

■ メモリスイッチ10H

ビット	意　味
bit7	メモリスイッチの設定と初期化 （1→する／0→しない）[LT・HA]

■ メモリスイッチ14H

モデム設定（1）[HA]

ビット	意　味
bit7	ブザー停止（1→する／0→しない）
bit6～5	ダイヤルモード （11B→20PPS／10B→10PPS／00B→トーン）
bit4	ループバックテスト（1→しない／0→する）
bit3～2	通信速度（11B→AUTO／10B→1200／01B→300／00B→2400）
bit1～0	モデムモード （10B→CALL／01B→ANS／00B→AUTO）

■ メモリスイッチ15H

モデム設定（2）[HA]

ビット	意　味
bit5	ロスオブキャリアディスコネクト （1→しない／0→する）
bit4	CIオプション選択（1→しない／0→する）
bit3	CDオプション選択（1→する／0→しない）
bit2	アンサートーン確認（1→しない／0→する）
bit1	アンサートーン送出（1→しない／0→する）
bit0	自動着信（1→しない／0→する）

■ メモリスイッチ18H

無操作時アラーム時間 [LT・HA]

ここで指定した時間（単位：分）キー入力がないときブザーを鳴らす。

0を設定するとこの機能を停止する。

■ メモリスイッチ19H

オートパワーオフ時間 [LT・HA]

ここで指定した時間（単位：分）キー入力がないとき電源をOFFにする。

0を設定するとこの機能を停止する。

HAではMSW 21Hのbit1＝0の場合次回電源ON時にこの内容は0になる。

■ メモリスイッチ20H モード設定、内蔵モデムの設定

HANDYメニュー [HA]

ビット	意　味
bit5～4	内蔵モデム （10B→AT／01B→V.25bis／00B→使用しない）
bit3	RAMドライブ書き込み禁止 （1→する／0→しない）
bit2	RAMドライブからの起動 （1→する／0→しない）
bit1～0	RAMドライブ用メモリの使用 （10B→しない／01B→増設メモリ／00B→RAMドライブ）

■ メモリスイッチ21H

HANDYメニュー 動作環境の設定 [HA]

ビット	意　味
bit3	キーボード設定 （1→トグル／0→ノーマル）
bit1	オートパワーオフ機能 （1→使用する／0→使用しない）
bit0	レジューム機能 （1→使用する／0→使用しない）

I/Oポート一覧

PC–9801でデバイスにアクセスするためのI/Oポートの一覧である。各ポートは種類ごとにまとめて掲載した。各ポートの詳細は『テクニカルデータブック』に譲る。

■ 割り込みコントローラ（マスタ）［ノーマル・ハイレゾ］ 8259A

I/Oポート	意　味
0000H	[READ] IRR/ISRリード、ポーリングデータリード [WRITE] ICW1/OCW2/OCW3ライト
0002H	[READ] IMRリード [WRITE] IMRライト（ICW2/ICW3/ICW4ライト） • IMR/ISR/IRRのビット割り当て ビット　意　味 bit7　(INT 0FH)スレーブPIC、マスタPIC不完全割り込み bit6　(INT 0EH)拡張バスINT2端子 bit5　(INT 0DH)拡張バスINT1端子 bit4　(INT 0CH)RS–232C(8251A) bit3　(INT 0BH)拡張バスINT0端子 bit2　(INT 0AH)VSYNC bit1　(INT 09H)キーボード(8251A) bit0　(INT 08H)タイマ(8253A CH#0) （IMR＝1→割り込みマスク／ IMR＝0→割り込み許可） （ISR＝1→割り込みサービス中） （IRR＝1→割り込み要求あり）

■ 割り込みコントローラ［LT・HA］ V50内蔵ICU（8259Aサブセット）

I/Oポート	意　味
0000H	[READ] IRR/ISRリード、ポーリングデータリード [WRITE] ICW1/OCW2/OCW3ライト
0002H	[READ] IMRリード [WRITE] IMRライト（ICW2/ICW3/ICW4ライト） • IMR/ISR/IRRのビット割り当て ビット　意　味 bit7　(INT 0FH)PIC不完全割り込み bit6　(INT 0EH)フロッピーディスクドライブ bit5　(INT 0DH)未使用 bit4　(INT 0CH)RS–232C(8251A) bit3　(INT 0BH)未使用 bit2　(INT 0AH)V50内蔵タイマチャネル1 bit1　(INT 09H)キーボード(8251A) bit0　(INT 08H)V50内蔵タイマチャネル0 （IMR＝1→割り込みマスク／ IMR＝0→割り込み許可） （ISR＝1→割り込みサービス中） （IRR＝1→割り込み要求あり）

■ 割り込みコントローラ（スレーブ）［ノーマル・ハイレゾ］ 8259A

I/Oポート	意　味
0008H	[READ] IRR/ISRリード、ポーリングデータリード [WRITE] ICW1/OCW2/OCW3ライト
000AH	[READ] IMRリード [WRITE] IMRライト（ICW2/ICW3/ICW4ライト） • IMR/ISR/IRRのビット割り当て ビット　意　味 bit7　(INT 17H)不完全割り込み bit6　(INT 16H)NDP［8086・V30］ なし［ノーマル:286・386・486］ プリンタポート PORT C3 ［フルセントロニクス］ bit5　(INT 15H)拡張バスINT6端子 bit4　(INT 14H)拡張バスINT5端子 bit3　(INT 13H)拡張バスINT42端子 bit2　(INT 12H)拡張バスINT41端子 bit1　(INT 11H)拡張バスINT3端子 bit0　(INT 10H)プリンタポート PORT C3 ［ノーマル:8086・V30］ なし［ノーマル：286・386・486・ハイレゾ］ （IMR＝1→割り込みマスク／ IMR＝0→割り込み許可） （ISR＝1→割り込みサービス中） （IRR＝1→割り込み要求あり）

■ DMAコントローラ［ノーマル・ハイレゾ］ 8237A

I/Oポート	意　味
0001H	[READ/WRITE]　チャネル0アドレス
0003H	[READ/WRITE]　チャネル0カウント
0005H	[READ/WRITE]　チャネル1アドレス
0007H	[READ/WRITE]　チャネル1カウント
0009H	[READ/WRITE]　チャネル2アドレス
000BH	[READ/WRITE]　チャネル2カウント
000DH	[READ/WRITE]　チャネル3アドレス
000FH	[READ/WRITE]　チャネル3カウント
0011H	[READ]　リードステータス ビット　意　味 bit7～4　RQ3～RQ0（チャネル3～0がリクエスト） bit3～0　TC3～TC0（チャネル3～0がTCに到達） [WRITE]　ライトコマンド ビット　意　味 bit7　KS（DACKアクティブ） (1→HIGH／0→LOW) bit6　DS（DREQアクティブ） (1→LOW／0→HIGH) bit5　WS（ライト選択）（TM＝1のとき無視） (1→拡張ライト選択／ 0→遅れライト選択) bit4　PR（優先順位） (1→回転優先順位／0→固定優先順位) bit3　TM（タイミング）（MM＝1のとき無視） (1→圧縮タイミング／0→通常タイミング) bit2　CE（コントローラ） (1→禁止／0→許可) bit1　AH（チャネル0アドレスホールド） (MM＝0のとき無視) (1→許可／0→禁止) bit0　MM（メモリ→メモリ転送） (1→許可／0→禁止)
0013H	[WRITE]　ライトリクエスト ビット　意　味 bit7～3　—— bit2　RB bit1～0　CS1、CS0（DMAチャネル）
0015H	[WRITE]　ライトシングルマスクレジスタビット ビット　意　味 bit7～3　—— bit2　MK（マスク）（1→セット／0→クリア） bit1～0　CS1、CS0（DMAチャネル選択） (11B→チャネル3／10B→チャネル2／ 01B→チャネル1／00B→チャネル0)
0017H	[WRITE]　ライトモード ビット　意　味 bit7～6　MS1、MS0（モード選択） 01B→シングルモード（PC–9800では他のモードの動作は保証されない）

I/Oポート	意味
	bit5　ID（アドレス・インクリメント／デクリメント） (1→デクリメント／0→インクリメント) bit4　AT（オートイニシャライズ） (1→許可／0→禁止) bit3〜2　TR1、TR0（転送モード）11B→禁止／10B→リード転送（I/O→メモリ）／01B→ライト転送（メモリ→I/O）／00B→ベリファイ転送 bit1〜0　CS1、CS0（DMAチャネル選択） (11B→チャネル3／10B→チャネル2／01B→チャネル1／00B→チャネル0)
0019H	[WRITE]　クリアバイトポインタフリップフロップ
001BH	[READ]　リードテンポラリレジスタ [WRITE]　マスタクリア
001DH	[WRITE]　クリアマスクレジスタ ビット　意味 bit7〜3　—— bit2　MK（マスク）（1→セット／0→クリア） bit1〜0　CS1、CS0（DMAチャネル選択）
001FH	[WRITE]　ライトオールマスクレジスタビット ビット　意味 bit7〜4　—— bit3〜0　MB3〜MB0（チャネル3〜0のマスク） (1→セット／0→クリア)

■ DMAバンク［ノーマル・ハイレゾ］

I/Oポート	意味
0021H	[WRITE]　チャネル1バンク（A23〜A16）
0023H	[WRITE]　チャネル2バンク（A23〜A16）
0025H	[WRITE]　チャネル3バンク（A23〜A16）
0027H	[WRITE]　チャネル0バンク（A23〜A16）

*V30機ではA19〜A16のみ有効
*XAではA22〜A16のみ有効

I/Oポート	意味
0029H	[READ]　なし [WRITE]　バンクアドレスオートインクリメントモードレジスタ ビット　意味 bit7〜4　0 bit3〜2　M1、M0 インクリメントのDMA境界 11B→16Mバイト（XAは設定不可）／10B→設定不可／01B→1Mバイト／00B→64Kバイト V30機には存在しない bit1〜0　CS1、CS0（DMAチャネル選択）

■ V50内蔵DMAユニット［LT・HA］

I/Oポート	意味
00E0H	[READ/WRITE]　DICM
00E1H	[READ/WRITE]　DCH
00E2〜00E3H	[READ/WRITE]　DBC/DCC
00E4〜00E6H	[READ/WRITE]　DBA/DCA
00E8〜00E9H	[READ/WRITE]　DDC
00EAH	[READ/WRITE]　DMD
00EBH	[READ/WRITE]　DST
00EFH	[READ/WRITE]　DMK

■ システムポート［ノーマル・ハイレゾ・LT・HA］8255A

I/Oポート	意味
0031H	PORT A（モード0入力） *ハードウェア的にはDIP SW 2が直接ポートに接続されている *XAはDIP SW 1-1〜1-8の状態 *NOTEは、bit7、6、4のみ使用 *E・F・Mはbit4〜0のみ使用 *ハイレゾモードはbit4のみ使用 *LT・HAはE3Hを示す [READ]　PORT Aリード (DIP SW 2の状態：1→OFF／0→ON) ビット　意味 bit7　GDC CLOCK（DIP SW 2-8） (1→2.5MHz／0→5MHz) bit6　フロッピーディスクモータ（DIP SW 2-7） [LV21・CV21・UV11] (1→制御しない／0→制御する) vfキー [NUMキーとvfキーの両方を持つ機種] (1→有効／0→無効) 起動時のFD#3、#4認識 [上記以外のデスクトップ] (1→する／0→しない) bit5　SASIハードディスク（DIP SW 2-6） (1→BIOSが認識する／0→BIOSが認識しない) bit4　メモリスイッチ（DIP SW 2-5） (1→リセット時初期化／0→保持) bit3　起動時のテキスト画面行数（DIP SW 2-4） (1→20行／0→25行) bit2　起動時のテキスト画面桁数（DIP SW 2-3） (1→40文字／0→80文字) bit1　ROMモード（DIP SW 2-2） (1→ROM BASIC起動／0→ターミナルモード起動) bit0　常にOFF（DIP SW 2-1） [WRITE]　PORT Aライト（無動作）
0033H	PORT B（モード0入力） H98では、NMIマスク状態のときにこのポートをリードすると、パリティエラーによりNMIリクエストがクリアされる。 [READ]　PORT Bリード ビット　意味 bit7　$\overline{\text{CI}}$（RS-232CのCI信号） (PC-9801初代ではN.C.) bit6　$\overline{\text{CS}}$（RS-232CのCS信号） bit5　$\overline{\text{CD}}$（RS-232CのCD信号） bit4　INT3（拡張バスのINT 3割り込み入力端子の状態）［ノーマル］ SHUT 0（システムポートPORT C7に接続）［ハイレゾ］ bit3　CRTT（CRTタイプ）（DIP SW 1-1） ［ノーマル］1→高解像度(ON)（ハイレゾ兼用機・NESA・LAPTOP・NOTE・LT・HAでは1に固定）／0→標準解像度(OFF) SHUT 1（システムポートPORT C7に接続）［ハイレゾ］ bit2　IMCK（内部RAMパリティエラー） bit1　EMCK（外部RAMパリティエラー） bit0　CDAT（カレンダ時計の読み出しデータ） [WRITE]　PORT Bライト（無動作）
0035H	PORT C（モード0出力） [READ/WRITE]　PORT Cリード、ライト ビット　意味 bit7　SHUT0（シャットダウンフラグ0）(*1)

I/Oポート	意　味
	bit6　PSTBM（プリンタのPSTB信号のマスク） （1→マスクする／0→マスクしない） PC-9801初代では未使用（0094H bit4で制御）
	bit5　SHUT1（シャットダウンフラグ1）(*1)
	bit4　MCHKEN（メモリチェック）（1→エラーチェック有効／0→エラーチェック無効） PC-9801U2では未使用。H98では常に1と同等の動作
	bit3　BUZ（ブザー）（1→停止／0→鳴動）
	bit2　TXRE（RS-232CのTxRDYによる割り込み要求のマスク）（1→許可／0→禁止）
	bit1　TXEE（RS-232CのTxEMPTYによる割り込み要求のマスク） （1→許可／0→禁止）
	bit0　RXRE（RS-232CのRxRDYによる割り込み要求のマスク）（1→許可／0→禁止）
	*1:286・386・486搭載機のみ使用。 SHUT0　SHUT1　動作 1　1　リセット処理ルーチン実行 0　x　CPUリセット後、プログラム実行を継続 SS←[0000:0406H] SP←[0000:0404H] RETF
0037H	[WRITE]　コントロールレジスタライト

■ タイマ［ノーマル・ハイレゾ・LT・HA］8253A

(LT・HAはV50内蔵TCU)

I/Oポート	意　味
0071H	カウンタ0（モード3） インターバルタイマ（INT 08H）用 [READ]　カウンタ#0をリード [WRITE]　カウンタ#0へのロード
0073H	カウンタ1（モード3） [初代・E・F・M・LT・HA・ハイレゾ]
3FDBH	カウンタ1（モード3）[上記以外のノーマル] PC-9801初代・E・F・Mではメモリリフレッシュ用。その他のノーマルではスピーカー周波数設定用（0073Hではアクセスできない） LT・HAではINT 0AH割り込み用 ハイレゾではスピーカー周波数設定用 [READ]　カウンタ#1をリード [WRITE]　カウンタ#1へのロード
0075H	カウンタ2（モード3） RS-232Cのボーレート生成用 [READ]　カウンタ#2をリード [WRITE]　カウンタ#2へのロード
0077H	コントロールレジスタ
3FDFH	コントロールレジスタ 初代・E・F・M以外のノーマルは、0077Hと3FDFHで同じポートにアクセスできる [READ]　なし [WRITE]　モード指定 ビット　意　味 bit7～6　SC1、SC0（セレクトカウンタ） （11B→使用不可／10B→カウンタ#2／01B→カウンタ#1／00B→カウンタ#0） bit5～4　RL1、RL0（リード、ロード） （11B→LSB、MSBの順にリード、ロード／10B→MSBのリード、ロード／01B→LSBのリード、ロード／00B→カウントラッチ）

I/Oポート	意　味
	bit3～1　M2～M0（モード） x11B→モード3（方形波ジェネレータ）／ x10B→モード2（レートジェネレータ）／ 000B→モード0（カウント終了割り込み） bit0　BCD（1→BCDカウント／0→バイナリカウント）

■ カレンダ時計［ノーマル・ハイレゾ・LT・HA］μPD4990、μPD1990

I/Oポート	意　味
0020H	[WRITE]　セットレジスタ ビット　意　味 bit7　―― bit6　―― bit5　データインプット bit4　クロック bit3　ストローブ bit2　コマンドコードC2 bit1　コマンドコードC1 bit0　コマンドコードC0

μPD4990/1990のデータアウト端子はシステムポートB0に接続

■ NMIコントロール［ノーマル・ハイレゾ］

I/Oポート	意　味
0050H	NMIフリップフロップ・リセット [WRITE]　任意の値を書き込むとNMIを無効にする
0052H	NMIフリップフロップ・セット [WRITE]　任意の値を書き込むとNMIを無効にする

*PC-9801U2・LT・HAにはこの機能がない

■ メモリウィンドウ［ハイレゾ］

I/Oポート	意　味
0091H	08、09バンク アドレスの上位8ビット(bit23～bit17)を本ポートのbit7～1に出力する。bit0の値は無視される。80000～9FFFFHに指定されたアドレスのメモリがマッピングされる [WRITE]　RAMウィンドウ・マッピング設定
0093H	0A、0Bバンク アドレスの上位8ビット(bit23～bit17)を本ポートのbit7～1に出力する。bit0の値は無視される。A0000～BFFFFHに指定されたアドレスのメモリがマッピングされる。 [WRITE]　RAMウィンドウ・マッピング設定

■ タイムスタンパ［H98］

307.2kHzでカウントアップする24ビットバイナリカウンタのカウント値を示す。

I/Oポート	意　味
005C～005DH	高分解能ARTICポート [READ]　カウンタのbit15～0 (分解能:約3.26μ秒、最大値:約213ミリ秒)
005E～005FH	低分解能ARTICポート [READ]　カウンタのbit23～8 (分解能:約833μ秒、最大値:約54.6秒) [WRITE]　0.6μ秒ウェイト

■ ウェイト［H98］

I/Oポート	意味
005FH	［READ］ タイムスタンパ読み出し ［WRITE］ ハードウェアで約 0.6 μ秒のウェイトがかかる

■ キーボードインターフェイス［ノーマル・ハイレゾ・LT・HA］ 8251A

I/Oポート	意味
0041H	［READ］ 受信データレジスタ ［WRITE］ 送信データレジスタ
0043H	［READ］ ステータスリード ビット 意味 bit7 KB コネクタ (6)（1→HIGH／0→LOW） bit6 —— bit5 フレーミングエラー検出（1→あり／0→なし） bit4 基本的に発生しないはず bit3 パリティエラー検出（1→あり／0→なし） bit2 TxEMP（送信終了） bit1 RxRDY（データ受信） bit0 TxRDY（送信可能） ［WRITE］ モードセット、コマンドワードライト ●コマンドワード ビット 意味 bit7 —— bit6 リセット（1→する／0→しない） bit5 RTS 信号（1→$\overline{RDY}$信号は常にインアクティブ／0→$\overline{RDY}$信号は RxRDY の状態に従う） bit4 エラーフラグクリア（1→8251A の FE、OE、PE をクリアする／0→エラークリアしない） bit3 TxDATA ブレーク送信 1→$\overline{RST}$端子を LOW レベルにする（キーボードをリセットする）／0→$\overline{RST}$端子を HIGH レベルにする bit2 受信許可（1→キーボードからデータ受信する／0→キーボードからデータ受信しない） bit1 DTR 信号 1→$\overline{RTY}$信号を HIGH（インアクティブ）にする／0→$\overline{RTY}$信号を LOW（アクティブ）にする キーボードにデータの再送を要求する bit0 送信許可（1→$\overline{RST}$端子からデータ送信する／0→$\overline{RST}$端子からデータ送信しない）

■ マウス［ノーマル］ 8255A

I/Oポート	意味
7FD9H	PORT A（モード 0 入力） ［READ］ PORT A リード（マウスの状態） ビット 意味 bit7 LEFT（マウスの左ボタンの状態）（1→離されている／0→押されている） bit6 MID（マウスの中ボタンの状態）（1→離されている／0→押されている） bit5 RIGHT（マウスの右ボタンの状態）（1→離されている／0→押されている） bit4 —— bit3～0 MD3～MD0（SXY、SHL で選択されたマウスのデータ）
7FDBH	PORT B（モード 0 入力） ［READ］ PORT B リード ビット 意味 bit7 —— RL のノーマルモードでは、ハイレゾモードのように DIP SW 1-4 は読めない bit6 RAMKL DIP SW 3-6 の設定状態（1→バンク 8、9 の内部 RAM を使用する／0→バンク 8、9 の内部 RAM を切り離す） RL でも読み出せるが、内部 RAM は切り離されない bit5～2 —— bit1 SPDSW 286 クロック切り換え SW の状態 ［RX2、RX21、EX2、LX2、DX2］（1→12MHz／0→10MHz） bit0 ——
7FDDH	PORT C（bit7～4←モード 0 出力、bit3～0←モード 0 入力） ［READ］ PORT C リード ビット 意味 bit7 HC 0→1 になったときにカウント値がバッファにコピーされる bit6～5 SXY、SHL MD3～MD0 に出力するデータを選択（11B→Y 軸方向上位 4 ビットデータ／10B→Y 軸方向下位 4 ビットデータ／01B→X 軸方向上位 4 ビットデータ／00B→X 軸方向下位 4 ビットデータ） bit4 $\overline{INT}$ マウスのタイマ割り込み（1→マスク／0→発生） bit3 MODSW ノーマル、ハイレゾの設定状態 1→ノーマルモード（ノーマル・ハイレゾ両用機）／0→ノーマル専用機 bit2 CPUSW DIP SW 3-8 の設定状態 1→OFF(V30)／0→ON(80x86) bit1 SW1-6 RS-232C 同期モードの設定状態（1→ON／0→OFF） 初代・E・F・M・VM2・VF2・U2 にはない bit0 SW1-5 RS-232C 同期モードの設定状態（1→ON／0→OFF） 初代・E・F・M・VM2・VF2・U2 にはない。 ［WRITE］ PORT C ライト ビット 意味 bit7 HC 0→1 になったときにカウント値がバッファにコピーされる bit6～5 SXY、SHL MD3～MD0 に出力するデータを選択（11B→Y 軸方向上位 4 ビットデータ／10B→Y 軸方向下位 4 ビットデータ／01B→X 軸方向上位 4 ビットデータ／00B→X 軸方向下位 4 ビットデータ） bit4 $\overline{INT}$ マウスのタイマ割り込み要求のマスク（1→不許可／0→許可） bit3～0 0
7FDFH	［WRITE］ コントロールレジスタライト

■ マウス割り込みタイマ設定［ノーマル］

I/Oポート	意味
BFDBH	［WRITE］ 割り込みタイマ設定 ビット　意味 bit7〜2　0 bit1〜0　T1、T0（マウス割り込み周期） （11B→15Hz／10B→30Hz／01B→60Hz／00B→120Hz） PC-9871 マウスインターフェイスボードにはない。

■ マウス［ハイレゾ］8255A

I/Oポート	意味
0061H	PORT A（モード0入力） ［READ］ PORT Aリード（マウスの状態） ビット　意味 bit7　LEFT（マウスの左ボタンの状態）（1→離されている／0→押されている） bit6　MID（マウスの中ボタンの状態）（1→離されている／0→押されている） bit5　RIGHT（マウスの右ボタンの状態）（1→離されている／0→押されている） bit4　—— bit3〜0　MD3〜MD0（SXY、SHLで選択されたマウスのデータ） ［WRITE］ PORT Aライト（無動作）
0063H	PORT B（モード0入力） ［READ］ PORT Bリード ビット　意味 bit7　FDU外付け1Mバイトフロッピーディスクドライブのユニット番号（DIP SW 1-4、DIP SW 2-8［XA］） 1→OFF（外付けフロッピーディスクドライブ#3、#4）／0→ON（外付けフロッピーディスクドライブ#1、#2） bit6　DIP SW 3-6［RL］†、DIP SW 2-7［XA］（1→OFF／0→ON） bit5　DIP SW 2-6［XA］（1→OFF／0→ON）1［その他］ bit4　DIP SW 2-5［XA］（1→OFF／0→ON）1［その他］ bit3　DIP SW 2-4［XA］（1→OFF／0→ON）1［その他］ bit2　DIP SW 2-3［XA］（1→OFF／0→ON）1［その他］ bit1　DIP SW 2-2［XA］（1→OFF／0→ON）1［その他］ bit0　MOD81（ハイレゾモード時の8/10MHzの切り換えスイッチ）（1→8/16MHz／0→10/20MHz） DIP SW 2-1［XA］（1→OFF／0→ON:既定値） ［WRITE］ PORT Bライト（無動作）
0065H	PORT C （bit7〜4←モード0出力、bit3〜0←モード0入力） ［READ］ PORT Cリード ビット　意味 bit7　HC 0→1になったときにカウント値がバッファにコピーされる bit6〜5　SXY、SHL MD3〜MD0に出力するデータを選択 （11B→Y軸方向上位4ビットデータ／10B→Y軸方向下位4ビットデータ／01B→X軸方向上位4ビットデータ／00B→X軸方向下位4ビットデータ） bit4　$\overline{INT}$ マウスのタイマ割り込み要求のマスク（1→不許可／0→許可） bit3　MODSW ノーマル、ハイレゾの設定状態 0→ハイレゾモード（ノーマル・ハイレゾ両用機のみ） XAは常に0 bit2　0 bit1　RS-2 RS-232C同期モードの設定状態 XAではDIP SW 1-10 その他はDIP SW 1-6 （1→ON／0→OFF） bit0　RS-1 RS-232C同期モードの設定状態 XAではDIP SW 1-9 その他はDIP SW 1-5 （1→ON／0→OFF） ［WRITE］ PORT Cライト ビット　意味 bit7　0 bit6〜5　SXY、SHL MD3〜MD0に出力するデータを選択 （11B→Y軸方向上位4ビットデータ／10B→Y軸方向下位4ビットデータ／01B→X軸方向上位4ビットデータ／00B→X軸方向下位4ビットデータ） bit4　$\overline{INT}$ マウスのタイマ割り込み（1→マスク／0→発生） bit3〜0　0
0067H	［WRITE］ コントロールレジスタライト

■ シリアルポート［ノーマル・ハイレゾ・LT・HA］8251A

I/Oポート	意味
0030H	［READ］ 受信データレジスタ ［WRITE］ 送信データレジスタ
0032H	［READ］ ステータスリード ビット　意味 bit7　DSR信号（1→DSR ON／0→DSR OFF） bit6　同期検出（1→あり／0→なし） bit5　フレーミングエラー検出（1→あり／0→なし） bit4　オーバーランエラー検出（1→あり／0→なし） bit3　パリティエラー検出（1→あり／0→なし） bit2　TxEMP bit1　RxRDY bit0　TxRDY ［WRITE］ モードセット、コマンドワードライト ●コマンドワード ビット　意味 bit7　エンターハントフェーズ（同期モードのみ） bit6　リセット（1→する／0→しない） bit5　RTS信号（1→RTS ON／0→RTS OFF） bit4　エラーフラグクリア（1→する／0→しない） bit3　TxDATAブレーク送信（1→する／0→しない） bit2　受信許可（1→許可／0→禁止） bit1　DTR信号（1→DTR ON／0→DTR OFF） bit0　送信許可（1→許可／0→禁止）

■ 拡張 RS-232C 制御、チャネル 2 [PC-9861]

I/O ポート　意　味

00B0H
[READ]　リードシグナル
ビット　意　味
bit7　CI
bit6　CS
bit5　CD
bit4〜0　——
[READ]　割り込みレベルセンス
ビット　意　味
bit7〜2　——
bit1〜0　IR1、IR2
11B→INT3／10B→INT2／01B→INT1（ハイレゾモードの標準設定）／00B→INT0（出荷時設定）
[WRITE]　マスクセット
ビット　意　味
bit7〜3　——
bit2　TXR（RS-232C の TxRDY による割り込み要求のマスク）
bit1　TXE（RS-232C の TxEMPTY による割り込み要求のマスク）
bit0　RXR（RS-232C の RxRDY による割り込み要求のマスク）

■ 拡張 RS-232C 制御・チャネル 3 [PC-9861]

I/O ポート　意　味

00B2H
[READ]　リードシグナル
ビット　意　味
bit7　CI
bit6　CS
bit5　CD
bit4〜0　——
[READ]　割り込みレベルセンス
ビット　意　味
bit7〜2　——
bit1〜0　IR1、IR2
11B→INT6／10B→INT5（出荷時設定）／01B→INT4／00B→INT0（ハイレゾモードの標準設定）
[WRITE]　マスクセット
ビット　意　味
bit7〜3　——
bit2　TXR（RS-232C の TxRDY による割り込み要求のマスク）
bit1　TXE（RS-232C の TxEMPTY による割り込み要求のマスク）
bit0　RXR（RS-232C の RxRDY による割り込み要求のマスク）

■ 拡張 RS-232C インターフェイス・チャネル 2 [PC-9861] 8251A

I/O ポート　意　味

00B1H
[READ]　受信データレジスタ
[WRITE]　送信データレジスタ

00B3H
[READ]　ステータスリード
ビット　意　味
bit7　DSR 信号
(1→DSR ON／0→DSR OFF)
bit6　同期検出（1→あり／0→なし）
bit5　フレーミングエラー検出
(1→あり／0→なし)
bit4　オーバーランエラー検出
(1→あり／0→なし)
bit3　パリティエラー検出（1→あり／0→なし）
bit2　TxEMP
bit1　RxRDY
bit0　TxRDY

I/O ポート　意　味

[WRITE]　モードセット、コマンドワードライト
●コマンドワード
ビット　意　味
bit7　エンターハントフェーズ（同期モードのみ）
bit6　リセット（1→する／0→しない）
bit5　RTS 信号
(1→RTS ON／0→RTS OFF)
bit4　エラーフラグクリア
(1→する／0→しない)
bit3　TxDATA ブレーク送信
(1→する／0→しない)
bit2　受信許可（1→許可／0→禁止）
bit1　DTR 信号
(1→DTR ON／0→DTR OFF)
bit0　送信許可（1→許可／0→禁止）

■ 拡張 RS-232C インターフェイス・チャネル 3 [PC-9861] 8251A

I/O ポート　意　味

00B9H
[READ]　受信データレジスタ
[WRITE]　送信データレジスタ

00BBH
[READ]　ステータスリード
[WRITE]　モードセット、コマンドワードライト

■ プリンタポート [ノーマル・LT・HA] 8255A

I/O ポート　意　味

0040H
PORT A（モード 0 出力）
[READ/WRITE]　PORT A リード、ライト
プリンタポートに出力するパラレルデータ

0042H
PORT B（モード 0 入力）
[READ]　PORT B リード
ビット　意　味
bit7〜6　TYP1、0（システムタイプ）
(11B→PC-9801U2・LT・HA／10B→その他の機種／01B→未定義／00B→PC-9801)
bit5　MOD（システムクロック）
1→8MHz（タイマクロック 2.0MHz）／0→5/10MHz（タイマクロック 2.5MHz）
bit4　LCD プラズマディスプレイ使用状態 (DIP SW 1-3)
1→使用しない (DIP SW 1-3 OFF)、LT・HA／0→使用する (DIP SW 1-3 ON)
DIP SW の状態がそのまま見えるだけで、現在のモードを示しているわけではない。
ハイレゾ兼用機でも SW 読み出し・機能とも有効。
bit3　HGC グラフィック拡張機能 (DIP SW 1-8)
1→基本モード (DIP SW 1-8 OFF)、LT・HA／0→拡張モード (DIP SW 1-8 ON)
DIP SW の状態がそのまま見えるだけで、現在のモードを示しているわけではない。
bit2　プリンタインターフェイス $\overline{\text{BUSY}}$ 信号
(1→インアクティブ／0→アクティブ)
bit1　CPUT（動作 CPU [ノーマル]）
(1→V30・V33／0→8086・286・386・486)
実際に動作している CPU を示す。
システムタイプ [LT・HA] ‡
(1→HA／0→LT)
bit0　VF（VF フラグ）
(1→PC-9801VF2／0→その他の機種)
[WRITE]　PORT B ライト（無動作）

0044H
PORT C（モード 0 出力）
[READ]　PORT C リード
[WRITE]　PORT C ライト
ビット　意　味
bit7　$\overline{\text{PSTB}}$（プリンタストローブ信号）
bit6〜4　0
bit3　IR8(INT 10H) 割り込み要求 (*1)
bit2　0

I/O ポート	意　味
	bit1　RST287/RST387(*2)
	bit0　0

*1:8086・V30 動作時のみ。286・386・486 動作時は（V30 相当モードも含め）切り離される。LT・HA では未使用。
*2:287・387 動作時のみ。

0046H	[WRITE]　コントロールレジスタライト

■ プリンタポート［ハイレゾ］ 8255A

I/O ポート	意　味
0040H	PORT A（モード 1 出力） [READ/WRITE]　PORT A リード、ライト プリンタポートに出力するパラレルデータ
0042H	PORT B（モード 0 入力） [READ]　PORT B リード ビット　意　味 bit7　SELECT bit6　FAULT bit5　PE bit4　DC+5V bit3　INPUT BUSY bit2　BUSY bit1　NO USE bit0　ACK （1→インアクティブ／0→アクティブ） [WRITE]　PORT B ライト（無動作）
0044H	PORT C（bit7〜3→モード 1、bit2〜0→モード 0 出力） [READ/WRITE]　PORT C リード、ライト ビット　意　味 bit7　OBF bit6　ACK bit5〜4　—— bit3　INT 16H 出力（スレーブ PIC IR6） bit2　PSTB 出力 bit1　RST287/RST387 出力 bit0　INPUT PRIME 出力
0046H	[WRITE]　コントロールレジスタライト

■ プリンタインターフェイス拡張モードレジスタ［H98］

I/O ポート	意　味
0448H	[READ]　ステータスリード ビット　意　味 bit7　IS（INT ステータス） 1→INT 要因発生（このポートをリードすると 0 になる）／0→INT 要因なし bit6〜5　—— bit4　IE（INT イネーブル） （1→イネーブル／0→ディスエーブル） bit3　DMA（DMA/INT モード） （1→DMA モード／0→INT モード） bit2〜1　—— bit0　MOD（プリンタモード） 1→拡張モード（フルセントロニクスモード、DMA・DRST を有効にする）／ 0→互換モード（簡易セントロニクスモード、DMA、DRST を無効にする） [WRITE]　ステータスセット ビット　意　味 bit7　0 bit6　DRST（DMA TC 割り込みのリセット） （1→DMATC 割り込みのリセット／ 0→No Operation） bit5　0 bit4　IE INT イネーブル（割り込みがエッジモードのときは無効） （1→イネーブル／0→ディスエーブル） bit3　DMA（DMA/INT モード設定） （1→DMA モード／0→INT モード） bit2〜1　0 bit0　MOD（プリンタモード設定） 1→拡張モード（フルセントロニクスモード、DMA・DRST を有効にする）／ 0→互換モード（簡易セントロニクスモード、DMA・DRST を無効にする）

■ テキスト GDC［ノーマル・ハイレゾ］ μ PD7220

I/O ポート	意　味
0060H	[READ]　GDC ステータスリード ビット　意　味 bit7　LIGHT PEN DETECT bit6　HORIZONTAL BLANK bit5　VERTICAL SYNC bit4　DMA EXECUTE bit3　DRAWING bit2　FIFO EMPTY bit1　FIFO FULL bit0　DATA READY [WRITE]　GDC パラメータライト
0062H	[READ]　GDC データリード [WRITE]　GDC コマンドライト

■ VSYNC 割り込みトリガ［ノーマル・ハイレゾ］

I/O ポート	意　味
0064H	[WRITE]　任意の値を出力すると、つぎの VSYNC 開始時に INT 0AH 割り込みを 1 回発生する

■ モードレジスタ

I/O ポート	意　味
0068H	[WRITE]　ライトモードレジスタ 1 0000000*n*　ATR7［ノーマル］ （*n*=1(01H)→簡易グラフ／ *n*=0(00H)→バーティカルライン） 0000001*n*　グラフィック［ノーマル］ （*n*=1(03H)→モノクロ／ *n*=0(02H)→カラー） 0000010*n*　桁数［ノーマル］ （*n*=1(05H)→40 字／ *n*=0(04H)→80 字） 0000011*n*　文字フォント［ノーマル］ （*n*=1(07H)→7×13／ *n*=0(06H)→6×8） 0000100*n*　ラインモード［ノーマル］ （*n*=1(09H)→200 ライン／ *n*=0(08H)→400 ライン） 0000101*n*　漢字アクセス［ノーマル・ハイレゾ］ （*n*=1(0BH)→ドットアクセス／ *n*=0(0AH)→コードアクセス） 0000110*n*　不揮発メモリ［ノーマル・ハイレゾ］ （*n*=1(0DH)→ライト可／ *n*=0(0CH)→ライト不可） 0000111*n*　画面表示（テキスト、グラフィック）［ノーマル・ハイレゾ］ （*n*=1(0FH)→する／ *n*=0(0EH)→しない）
006AH	[WRITE]　ライトモードレジスタ 2 0000000*n*　色数［ノーマル］ （*n*=1(01H)→16 色／ *n*=0(00H)→8 色） 0000010*n*　EGC モード（*1） （*n*=1(05H)→拡張モード／ *n*=0(04H)→互換モード）

I/Oポート	意味
	0000011n (*1) のフリップフロップ変更 (n=1(07H) →許可/ n=0(06H) →禁止)
	0010000n 多色表示モード (*1) [H98] (n=1(21H) → 多色拡張モード/ n=0(20H) →標準モード) 多色拡張モードのときのみ、(*2) が有効。
	0010001n 表示色設定 (*1、*2) [H98] (n=1(23H) → 64K 色/64K 色/ n=0(22H) → 256 色/16M 色)
	0010010n 64K 色表示パレット (*1、*2) [H98] (n=1(25H) →一部パレット/ n=0(24H) →パレット未使用) 表示色 64K モードのときのみ有効。
	0010011n 全画面リバース (*1) [H98] (n=1(27H) → ON/ n=0(26H) → OFF)
	0010101n 256 色パレットレジスタ高速書き込み (*1) [H98] (n=1(2BH) → ON/ n=0(2AH) → OFF)
	0010110n 256 色オーバースキャンカラー (*1、*2) [H98] (n=1(2DH) →使用/ n=0(2CH) →未使用)
	0100000n テキスト、グラフィック表示モード [ノーマル] (n=1(41H) →プラズマディスプレイモード/ n=0(40H) → CRT モード) CRT モードではテキストがグラフィックより 1 ドット左にはみ出す。 プラズマディスプレイモードではズレは発生しない。 ラップトップ・NOTE は常にプラズマディスプレイモード。
	0110000n E^2GC モード (*1) [H98] (n=1(61H) →拡張 EGC/ n=0(60H) → EGC 互換) EGC 拡張モード移行は CGmode = 1 でなければならない。 EGC 互換モード移行は 04A0H の EGC レジスタが FFF0H でなければならない。
	0110001n VRAM 構成 (*1、*2) [H98] (n=1(63H) →パックト/ n=0(62H) →プレーン)
	0110011n 描画プロセッサ (*1) [H98] (n=1(67H) → AGDC 使用/ n=0(66H) → GDC 使用)
	1000001n GDC CLOCK-1 [ノーマル] (n=1(83H) → 2.5MHz/ n=0(82H) → 5MHz) GDC クロックを 5MHz にするには、GDC CLOCK-2 も 5MHz に設定しなければならない。しかし、GDC クロックを 2.5MHz にするには GDC CLOCK-1 または 2 のどちらか一方の設定でよい。
	1000010n GDC CLOCK-2 [ノーマル] (n=1(85H) → 2.5MHz/ n=0(84H) → 5MHz)
	1100010n グラフ GDC 動作モード (*1) [H98] (n=1(C5H) →マスタ動作/ n=0(C4H) →スレーブ動作)

■ ボーダーカラー [ノーマル]

I/Oポート	意味
006CH	[WRITE] ビット 意味 bit7 0 bit6〜4 カラーコード bit3〜0 0

標準ディスプレイ使用時のみ設定可能

■ テキスト表示ラインカウンタ制御回路 [ノーマル・ハイレゾ]

I/Oポート	意味
0070H	[WRITE] キャラクタ位置ライン数
0072H	[WRITE] ボディーフェイスライン数
0074H	[WRITE] キャラクタライン数
0076H	[WRITE] スムーススクロールライン数
0078H	[WRITE] スクロールエリア上辺位置行数
007AH	[WRITE] スクロールエリア行数

■ 漢字 ROM [ノーマル]

I/Oポート	意味
00A1H	[WRITE] ライト文字コード第 2 バイト
00A3H	[WRITE] ライト文字コード第 1 バイト
00A5H	[WRITE] ライトラインカウンタ
00A9H	[READ] リード文字パターン [WRITE] ライト文字パターン

■ フォントサイズの切り替え [H98 ハイレゾ] ‡

I/Oポート	意味
00A7H	[WRITE] (00010011(13H) → 24 ドット/ 00010010(12H) → 16 ドット)

■ グラフィック GDC [ノーマル・ハイレゾ] μ PD7220

I/Oポート	意味
00A0H	[READ] GDC ステータスリード ビット 意味 bit7 LIGHT PEN DETECT (常に 1 を示す) bit6 HORIZONTAL BLANK bit5 VERTICAL SYNC bit4 DMA EXECUTE bit3 DRAWING bit2 FIFO EMPTY bit1 FIFO FULL bit0 DATA READY [WRITE] GDC パラメータライト
00A2H	[READ] GDC データリード [WRITE] GDC コマンドライト
00A4H	表示画面選択レジスタ [ノーマル] [WRITE] プレーン選択 (00B →プレーン 0 / 01B →プレーン 1)
00A6H	描画画面選択レジスタ [ノーマル] [WRITE] プレーン選択 (00B →プレーン 0 / 01B →プレーン 1)

■ パレットレジスタ

I/Oポート	意味
00A8H	[READ] なし [H98 以外] /パレットデータ [H98] [WRITE] パレットレジスタ (#3/#7) [ノーマル 8 色] [WRITE] パレットアドレス [ノーマル 16 色・ハイレゾ]

I/Oポート	意味
00AAH	[READ] なし[H98以外]/パレットデータ[H98] [WRITE] パレットレジスタ(#1/#5)[ノーマル8色] [WRITE] パレット GREEN[ノーマル16色・ハイレゾ]
00ACH	[READ] なし[H98以外]/パレットデータ[H98] [WRITE] パレットレジスタ(#2/#6)[ノーマル8色] [WRITE] パレット RED[ノーマル16色・ハイレゾ]
00AEH	[READ] なし[H98以外]/パレットデータ[H98] [WRITE] パレットレジスタ(#0/#4)[ノーマル8色] [WRITE] パレット BLUE[ノーマル16色・ハイレゾ]

H98シリーズのみ、以下のようにパレットデータをリードできる。

8色モード時	リード・ライトは常時可能。リード・ライトは順序に関係なく実行可能
16色モード時	リードはBLANK中に行う。ライトは常時可能。リード時はパレットNO指定後、G/R/B順で読み出す
256色モード時	リードはBLANK中に行う。リード時はパレットNO指定後、G/R/B順で読み出す。ライト時には一瞬画面が乱れることがある

■ 256色パレットライトステータスフラグ[H98]

I/Oポート	意味
09A2H	[READ] bit7～1 — bit0 ST 256色パレット高速書き込みOFFのとき、このビットが0であることを確認してから、値を設定しなければならない。値を設定することで、STは1になり、最大41μ秒続く。

■ ピクセルマスク[H98]

I/Oポート	意味
09AEH	[READ/WRITE] PM7～0(ライトピクセルマスク) ピクセルデータの該当するビットを0にすることによりマスクされる。通常、256色/16M色のときFFH、16色/16M色のとき0FHを設定する。

■ GRCG[ノーマル]

I/Oポート	意味
007CH	[WRITE] ライトモードレジスタ ビット 意味 bit7 CGmode(GRCGモードの選択)(1→有効にする/0→無効にする) bit6 MWmode(CPUのVRAMアクセスの選択)(1→ライト時RMWモード/0→リード時TCRモード、ライト時TDWモード) bit5～4 0 bit3～0 $\overline{\text{P3EN}}$～$\overline{\text{P0EN}}$(該当するプレーンの選択)(1→無効にする/0→有効にする)
007EH	[WRITE] タイルレジスタ(0～3)

■ GRCG[ハイレゾ]

I/Oポート	意味
00A4H	[WRITE] ライトモードレジスタ ビット 意味 bit7 CGmode(GRCGモードの選択)(1→有効にする/0→無効にする) bit6 RMWmode(CPUのVRAMアクセスの選択)(1→ライト時RMWモード/0→リード時TCRモード、ライト時TDWモード) bit5～4 RP1、RP2 CGモード=0のとき→00B/CGモード=1のとき→リード時のプレーン選択(P0～P3) bit3～0 $\overline{\text{P3EN}}$～$\overline{\text{P0EN}}$(該当するプレーンの選択)(1→無効にする/0→有効にする)
00A6H	[WRITE] ライトタイルレジスタ(0～3)

■ エンハンストグラフィックチャージャ(EGC)

I/Oポート	意味
04A0H	[WRITE] EGCレジスタ1ライト ビット 意味 bit15～4 1 bit3～0 $\overline{\text{P3EN}}$～$\overline{\text{P0EN}}$(アクティブプレーン)
04A2H	[WRITE] EGCレジスタ2ライト ビット 意味 bit15 0 bit14 FGC bit13 BGC bit12 0 bit11～8 リードプレーン bit7～0 1
04A4H	[WRITE] EGCレジスタ3ライト ビット 意味 bit15～14 0 bit13 コンペアリード bit12～11 ライトソース bit10 リードソース bit9～8 レジスタロード bit7～0 ROPレジスタ
04A6H	[WRITE] EGCレジスタ4ライト ビット 意味 bit15～0 0 FGC、BGC=1のとき ビット 意味 bit15～4 0 bit3～0 フォアグラウンドカラー
04A8H	[WRITE] EGCレジスタ5ライト ビット 意味 bit15～0 マスクレジスタ(1個のみ) FGC、BGC=1のとき ビット 意味 bit15～0 0
04AAH	[WRITE] EGCレジスタ6ライト ビット 意味 bit15～0 0 FGC、BGC=1のとき ビット 意味 bit15～4 0 bit3～0 バックグラウンドカラー
04ACH	[WRITE] EGCレジスタ7ライト ビット 意味 bit15～13 0 bit12 DIR bit11～8 0 bit7～4 ディスティネーションビットアドレス bit3～0 ソースビットアドレス
04AEH	[WRITE] EGCレジスタ8ライト(ビット長)

■ 1M バイトフロッピーディスク［ノーマル・ハイレゾ・PC−9801−15］μPD765A

I/O ポート	意　味
0090H	［READ］ ステータスレジスタ読み込み
0092H	［READ］ リザルトステータス読み込み ［WRITE］ コマンドレジスタ書き込み、パラメータ書き込み

■ 1M バイトフロッピーディスク・コントロールレジスタ［ノーマル・PC−9801−15］

I/O ポート	意　味
0094H	［READ］ リードスイッチ、シグナル

ビット	意　味
bit7	FINT1（インターフェイスボード上の DIP SW 1−7 の状態） 1 → OFF ／ 0 → ON（既定値。両用は 0 に固定）
bit6	FINT0（インターフェイスボード上の DIP SW 1−6 の状態） 1 → OFF（既定値。両用は 1 に固定）／ 0 → ON
bit5	DMACH（インターフェイスボード上の DIP SW 1−3 の状態） 1 → OFF ／ 0 → ON（既定値。両用は 0 に固定） PC−9801 初代では無効
bit3	TYP1（FDD #3/#4） （1 →両用／ 0 → 1M バイト専用） 両用インターフェイスで有効。専用インターフェイスは未定義
bit2	TYP0（FDD #1/#2） （1 →両用／ 0 → 1M バイト専用） 両用インターフェイスで有効。専用インターフェイスは未定義
bit1〜0	0

［WRITE］ ライトコントロールレジスタ

ビット	意　味
bit7	RST（μPD765A の RESET 端子への入力信号）
bit6	FRY（μPD765A の RDY 端子への入力信号）
bit5	——
bit4	DMAE（DMA 使用の選択） ［PC−9801 初代以外］ （1 → DMA モード／ 0 →プログラム I/O モード） プリンタの PSTB 信号のマスク［PC−9801 初代のみ］ 他機種では 0035H bit6 で制御。
bit3〜0	——

■ 内蔵 1M バイトフロッピーディスク・コントロールレジスタ［ハイレゾ］

I/O ポート	意　味
0094H	［READ］ リードスイッチ、シグナル

ビット	意　味
bit7	MODE（内蔵ドライブ） （1 →なし／ 0 →あり）
bit6	0
bit5	HD（アクセスモード） （1 → 640K バイトアクセスモード／ 0 → 1M バイトアクセスモード）
bit4	0
bit3〜0	——

［WRITE］ ライトコントロールレジスタ

ビット	意　味
bit7	RST（μPD765A の RESET 端子の入力信号となる）
bit6	FRY（μPD765A の RDY 端子の入力信号となる）
bit5	HD（アクセスモード） （1 → 640K バイトアクセスモード／ 0 → 1M バイトアクセスモード）
bit4	——
bit3	MTON（ディスクドライブの MOTOR ON 端子への入力信号） XL^2、RL 以降のハイレゾで有効。
bit2〜0	——

■ 内蔵両用 FD インターフェイスポート［ノーマル］

I/O ポート	意　味
00BEH	［READ］ リードモードステータス

ビット	意　味
bit7〜4	
bit3	DSW（DIP SW3−2 の状態） 1 → OFF（1M バイト指定）／ 0 → ON（640K バイト指定）
bit2	FIX（DIP SW3−1 の状態） 1 → ON（固定モード…PORT EXC 無効）／ 0 → OFF（自動切り換えモード…PORT EXC 有効）
bit1	FDD EXC（FDD Mode Exchange） （1 → 1M バイトアクセスモード／ 0 → 640K バイトアクセスモード）
bit0	PORT EXC（Port Exchange） （1 → 1M バイトインターフェイスモード／ 0 → 640K バイトインターフェイスモード）

［WRITE］ ライトモードチェンジ

ビット	意　味
bit7〜3	0
bit2	EMTON（ポート 0094H bit3(MTON) 有効、無効） 1 → 1M バイト インターフェイスモード時、常時モータ ON ／ 0 → 1M バイト インターフェイスモード時、MTON ビット有効（モータ制御あり） PC−9801UX 以降のみ有効。
bit1	FDD EXC（FDD Mode Exchange） （1 → 1M バイトアクセスモード／ 0 → 640K バイトアクセスモード）
bit0	PORT EXC（Port Exchange） （1 → 1M バイトインターフェイスモード／ 0 → 640K バイトインターフェイスモード）

■ 内蔵両用 FD インターフェイスポート［ハイレゾ］

I/O ポート	意　味
00BEH	［READ］ リードモードステータス

ビット	意　味
bit2	EMTON

［WRITE］ ライトモードチェンジ

ビット	意　味
bit7〜3	0
bit2	EMTON（ポート 0094H bit3(MTON) 有効、無効） 1 → 1M バイトインターフェイスモード時、常時モータ ON ／ 0 → 1M バイトインターフェイスモード時、MTON ビット有効（モータ制御あり） XL^2、RL 以降で有効。
bit1〜0	0

■ 640K バイトフロッピーディスク［ノーマル・LT・HA・PC−9801−09］μPD765A

I/O ポート	意　味
00C8H	［READ］ ステータスレジスタ読み込み
00CAH	［READ］ リザルトステータス読み込み ［WRITE］ コマンドレジスタ書き込み、パラメータ書き込み

■ 640K バイトフロッピーディスク・コントロールレジスタ

I/O ポート　意　味

00CCH　[READ]　リードスイッチ、シグナル
ビット　意　味
bit7　FINT1（インターフェイスボード上の DIP SW 1–7 の状態）
1 → OFF ／ 0 → ON（既定値。両用は 0 に固定）
bit6　FINT0（インターフェイスボード上の DIP SW 1–6 の状態）
1 → OFF（既定値。両用は 1 に固定）／ 0 → ON
bit5　DMACH（インターフェイスボード上の DIP SW 1–3 の状態）
1 → OFF（既定値。両用は 0 に固定）／ 0 → ON
bit4　RDY（ディスクドライブの Ready 端子の状態）
bit3　TYP1（FDD #3/#4）
（1 →両用／ 0 → 1M バイト専用）
両用インターフェイスで有効。専用インターフェイスは未定義。
bit2　TYP0（FDD #1/#2）
（1 →両用／ 0 → 1M バイト専用）
両用インターフェイスで有効。専用インターフェイスは未定義。
bit1〜0　0
両用インターフェイスで有効。専用インターフェイスは未定義。
[WRITE]　ライトコントロールレジスタ
ビット　意　味
bit7　RST（μ PD765A の RESET 端子への入力信号）
bit6　FRY（μ PD765A の RDY 端子への入力信号）
PC–9801U2・VF2・両用ドライブのみ有効。
bit5　AIE（アテンションインタラプト）
（1 → FRY のセット、リセットを無効／ 0 → FRY のセット、リセットを有効）
PC–9801U2・VF2・両用ドライブのみ有効。
bit4　DMAE（DMA 使用の選択）
（1 → DMA モード／ 0 →プログラム I/O モード）
bit3　MTON（ディスクドライブの MOTOR ON 端子への入力信号）
bit2　TMSK（μ PD765A からのタイマ割り込み要求のマスク）
（1 →タイマ割り込みを許可／ 0 →タイマ割り込みを不許可）
bit1　——
bit0　TTRG（VFO の TRIG IN 端子の入力信号）

■ 320K バイトフロッピーディスク [初代・E・F・M] 8255A

I/O ポート　意　味

0051H　PORT A
[READ]　PORT A リード
[WRITE]　PORT A ライト（無動作）

0053H　PORT B
[READ]　PORT B リード（診断用）
[WRITE]　PORT B ライト

0055H　PORT C
[READ]　PORT C リード
リードシグナル 2（一部診断用）
ビット　意　味
bit7　ATN コマンド（データ）送信要求
bit6　DAC コマンド（データ）受信完了
bit5　RFD コマンド（データ）受信準備完了
bit4　DAV コマンド（データ）送信完了
bit3　——
bit2　DAC コマンド（データ）受信完了
bit1　RFD コマンド（データ）受信準備完了
bit0　DAV コマンド（データ）送信完了

I/O ポート　意　味

[WRITE]　PORT C ライト（ライトシグナル 2）
ビット　意　味
bit7　ATN コマンド（データ）送信要求
bit6　DAC コマンド（データ）受信完了
bit5　RFD コマンド（データ）受信準備完了
bit4　DAV コマンド（データ）送信完了
bit3〜0　1
0057H　[WRITE]　コントロールレジスタライト

■ SASI タイプハードディスクインターフェイス [内蔵型・PC–9801–27]

I/O ポート　意　味

0080H　[WRITE]　データ入出力レジスタ

0082H　[READ]　ステータスレジスタ（NRDSW＝1 のとき）
ビット　意　味
bit7　REQ
bit6　ACK（内蔵タイプには存在しない）
bit5　BSY
bit4　MSG
bit3　CXD
bit2　IXO
bit1　——
bit0　INT
[READ]　ステータスレジスタ（NRDSW＝0 のとき）
ビット　意　味
bit7　CT0 SASI 1 台目
（インターフェイスボード DIP SW 1–1）
0 → ON（256 バイトセクタ）／
1 → OFF（512 バイトセクタ）
bit6　CT1 SASI 2 台目
（インターフェイスボード DIP SW 1–2）
0 → ON（256 バイトセクタ）／
1 → OFF（512 バイトセクタ）
bit5〜3　DT02、DT01、DT00 SASI1 台目容量（インターフェイスボード DIP SW 1–3〜5）
（111B →未接続／ 110B → 40M バイト／ 100B → 20M バイト／ 001B → 10M バイト／ 000B → 5M バイト）
bit2〜0　DT12、DT11、DT10（SASI 2 台目容量）
（インターフェイスボード DIP SW 1–6〜8）
（111B →未接続／ 110B → 40M バイト／ 100B → 20M バイト／ 001B → 10M バイト／ 000B → 5M バイト）
[WRITE]　コントロールレジスタ
ビット　意　味
bit7　CHEN（内部バスへのデータ出力）
（1 →許可／ 0 →禁止）
内蔵タイプには存在しない。
bit6　NRDSW（リード時のモード選択）
bit5　SEL 内部バスの SEL 信号制御
（1 → ON ／ 0 → OFF）
bit4　0
bit3　RST 1 → 0 にしたとき内部バスの RST 信号が 400 ナノ秒間 ON になる
bit2　0
bit1　DMAE（DMA 使用の選択）
（1 → DMA モード／ 0 →プログラム I/O モード）
bit0　INTE（割り込み要求のマスク）
（1 →許可／ 0 →不許可）

*NOTE の内蔵 IDE、H98 の ESDI は異なる。

■ SCSIインターフェイス［内蔵型・PC-9801-55］

I/Oポート　意味

0CC0H, 0CD0H, 0CE0H, 0CF0H　[READ]　補助ステータス

ビット	意味
bit7	INT
bit6	LCI
bit5	BSY
bit4	CIP
bit3〜2	0
bit1	PE
bit0	DBR

[WRITE]　AR（コントロールレジスタのアドレス）

0CC2H, 0CD2H, 0CE2H, 0CF2H　[READ]　コントロールレジスタ取得

- AR＝17H　SCSI Status
- AR＝30H　Memory Bank

ビット	意味
bit7	——
bit6	ROM0（ROMバンクセット）（1→上位ROMバンク／0→下位ROMバンク）
bit5〜4	——
bit3	MEM1（メモリアクセス可、不可）（1→ローカルメモリのシステム側からアクセス許可／0→ローカルメモリのシステム側からアクセス禁止）
bit2	IRE1（割り込み出力可、不可）（1→インターフェイスからシステム側へ割り込みを許可／0→インターフェイスからシステム側へ割り込みを禁止）
bit1	WRS1（SCSIバスのリセット）（1→SCSIのRST信号がインターフェイスボードへのI/Oによりアクティブ／0→インターフェイスボードからはリセットしてない）
bit0	——

- AR＝31H　Memory Window

ビット	意味
bit7	——
bit6〜5	HST1、HST0（本体タイプ設定値）
bit4〜0	WND4〜WND0（メモリウィンドウ設定値）

- AR＝32H　使用禁止（NECリザーブ）
- AR＝33H　RESET/INT

ビット	意味
bit7	RRST（SCSIバスのリセットピン状態）（1→SCSIバス上のRST信号が25μS以上LOWアクティブ。リードすると0になる。）
bit6	——
bit5〜3	ILV2〜ILV0（割り込みレベル）
bit2〜0	ID2〜ID0（インターフェイスボードのID番号）

- AR＝34H　使用禁止（NECリザーブ）

[WRITE]　コントロールレジスタ設定

- AR＝30H　Memory Bank

ビット	意味
bit7	0
bit6	ROM0（ROMバンクセット）（1→上位ROMバンク／0→下位ROMバンク）
bit5〜4	0
bit3	MEM1（メモリアクセス可、不可）（1→ローカルメモリのシステム側からアクセス許可／0→ローカルメモリのシステム側からアクセス禁止）
bit2	IRE1（割り込み出力可、不可）（1→インターフェイスからシステム側へ割り込みを許可／0→インターフェイスからシステム側へ割り込みを禁止）
bit1	WRS1（SCSIバスのリセット）（1→SCSIバス上RST信号をLOWアクティブ／0→インターフェイスボードからのバスリセットを解除）
bit0	0

- AR＝32H　使用禁止（NECリザーブ）
- AR＝33H　使用禁止（NECリザーブ）
- AR＝34H　使用禁止（NECリザーブ）
- AR＝35H　使用禁止（NECリザーブ）

[READ/WRITE]　コントロールレジスタ取得、設定　ただし*はREAD時1、WRITE時0とする。

- AR＝00H　Own ID

ビット	意味
bit7〜3	*
bit2〜0	ID2〜ID0（インターフェイスボードのID番号）

- AR＝01H　Control

ビット	意味
bit7	DMA
bit6	WDB
bit5〜2	*
bit1	HA
bit0	HPE

- AR＝02H　Time out Period
- AR＝03H　Total Sectors/CDB1
- AR＝04H　Total Head/CDB2
- AR＝05H　Total Cylinders(MSB)/CDB3
- AR＝06H　Total Cylinders(LSB)/CDB4
- AR＝07H　Logical Address(MSB)/CDB5
- AR＝08H　Logical Address /CDB6
- AR＝09H　Logical Address /CDB7
- AR＝0AH　Logical Address(LSB)/CDB8
- AR＝0BH　Sector Number/CDB9
- AR＝0CH　Head Number/CDB10
- AR＝0DH　Cylinder Number(MSB)/CDB11
- AR＝0EH　Cylinder Number(LSB)/CDB12
- AR＝0FH　Target Lun

ビット	意味
bit7〜3	*
bit2〜0	TL2〜TL0

- AR＝10H　Command Phase

ビット	意味
bit7	*
bit6〜0	CP6〜CP0

- AR＝11H　Synchronous Transfer

ビット	意味
bit7	*
bit6〜4	TP2〜TP0
bit3	*
bit2〜0	OF2〜OF0

- AR＝12H　Transfer Count(MSB)
- AR＝13H　Transfer Count
- AR＝14H　Transfer Count(LSB)
- AR＝15H　Destination I

ビット	意味
bit7〜3	*
bit2〜0	DI2〜DI0

- AR＝16H　Source ID

ビット	意味
bit7	ER
bit6	ES
bit5〜4	*
bit3	SIV
bit2〜0	SI2〜SI0

- AR＝18H　Command
- AR＝19H　Data

I/Oポート	意味
0CC4H, 0CD4H, 0CE4H, 0CF4H	[READ] ステータスリード ビット 意味 bit7 * bit6 TCI (DMATCO信号による割り込み) (1→割り込み発生／0→割り込みなし) bit5～2 * bit1～0 DMA1、DMA0 (使用するDMAチャネルのスイッチ状態) [WRITE] コマンドライト ビット 意味 bit7～5 0 bit4 TCIR bit3 TCMR bit2 TCMS bit1 DMER bit0 DMES インターフェイスボード上のDIP SWでI/Oアドレスが変更できる。

■ SCSIインターフェイスH98拡張モード[H98]

I/Oポート	意味
0EE0H	PORT 0CC0Hに準ずる
0EE2H	PORT 0CC2Hに準ずる
0EE4H	PORT 0CC4Hに準ずる

■ サウンドボード[PC-9801-26] YM2203

I/Oポート	意味
0188H	[READ] リードステータス ビット 意味 bit7 FM音源部BUSY bit6～2 —— bit1 FLAGB bit0 FLAGA [WRITE] ライトアドレス
018AH	[READ] リードデータ [WRITE] ライトデータ

■ GP-IBボード[PC-9801-29/K/N] μPD7210

I/Oポート	意味
0099H	[READ] リードスイッチ (ボード上のスイッチ) ビット 意味 bit7～6 GINT1、GINT2 (割り込みレベル) bit5 MS (モード指定) 1→マスタモード (OFF) ／ 0→スレーブモード (ON) bit4～0 MA4～MA0 (アドレス)
009BH	[READ] リードIFC (ボード上のレジスタ) ビット 意味 bit7 $\overline{\text{IFC}}$ (1→インアクティブ／0→アクティブ) bit6 GINT2 (リードスイッチと同じ) bit5 MS (リードスイッチと同じ) bit4～0 MA4～MA0 (リードスイッチと同じ)
00C1H	[READ] Data In [WRITE] Byte Out
00C3H	[READ] Interrupt Status1 ビット 意味 bit7 CPT bit6 APT bit5 DET bit4 END bit3 DEC bit2 ERR bit1 DO bit0 DI [WRITE] Interrupt Mask1 ビット 意味 bit7 CPT bit6 APT bit5 DET bit4 END bit3 DEC bit2 ERR bit1 DO bit0 DI
00C5H	[READ] Interrupt Status2 ビット 意味 bit7 INT bit6 SRQI bit5 LOK bit4 REM bit3 CO bit2 LOKC bit1 REMC bit0 ADSC [WRITE] Interrupt Mask2 ビット 意味 bit7 0 bit6 SRQI bit5 DMAI bit4 DMAO bit3 CO bit2 LOKC bit1 REMC bit0 ADSC
00C7H	[READ] Serial Poll Status ビット 意味 bit7 S8 bit6 PEND bit5～0 S6～S1 [WRITE] Serial Poll Mode ビット 意味 bit7 S8 bit6 rsv bit5～0 S6～S1
00C9H	[READ] Address Status ビット 意味 bit7 CIC bit6 $\overline{\text{ATN}}$ bit5 SPMS bit4 LPAS bit3 TPAS bit2 LA bit1 TA bit0 MJMN [WRITE] Address Mode ビット 意味 bit7 ton bit6 lon bit5～4 TRM1、TRM0 bit3～2 0 bit1～0 ADM1、ADM0
00CBH	[READ] Command Pass Through [WRITE] Auxiliary Mode ビット 意味 bit7～5 CNT2～CNT0 bit4～0 COM4～COM0

I/Oポート　意　味

00CDH　[READ]　Address0

ビット　意　味

bit7　――

bit6　DT0

bit5　DL0

bit4〜0　AD5-0〜AD1-0

[WRITE]　Address0/1

ビット　意　味

bit7　ARS

bit6　DT

bit5　DL

bit4〜0　AD5〜AD1

00CFH　[READ]　Address1

ビット　意　味

bit7　EOI

bit6　DT1

bit5　DL1

bit4〜0　AD5-1〜AD1-1

[WRITE]　End of String

■ EMSボード用制御ポート［PC-9801-53・98NOTE・98HA・etc］

I/Oポート	意　味	
08E1H	[WRITE]	C000H SEL BANK
08E3H	[WRITE]	C400H SEL BANK
08E5H	[WRITE]	C800H SEL BANK
08E7H	[WRITE]	CC00H SEL BANK
08E9H	[WRITE]	メモリアドレス（A23-A20）

■ マシンステータス読み出し、CPUシャットダウン［80286以上搭載機］†

I/Oポート　意　味

00F0H　[READ]　マシン・ステータス読み出し

ビット　意　味

bit7　未使用‡
起動状態では1［DA2・RA21・VX21・RL調べ］
DA2・RA21のROM内では参照なし

bit6　内蔵55タイプハードディスクドライブステータス†（1→内蔵55タイプハードディスクドライブなし／0→内蔵55タイプハードディスクドライブあり）

bit5　内蔵27タイプ(IDE)ハードディスクドライブステータス†（1→内蔵27タイプハードディスクドライブなし／0→内蔵27タイプハードディスクドライブあり）

bit4　不明‡
起動状態では［RA21-V30・VX21-80286・VX21-V30調べ］
起動状態では0［DA2-80386・RA21-80386の調べ］
DA2・RA21のROM内では参照なし

bit3　不明‡
起動状態では1［DA2・RA21・VX21調べ］
DA2・RA21のITF ROMで参照あり

bit2　リフレッシュモード‡（1→通常リフレッシュ／0→高速リフレッシュ）
VX21のV30モードでは、高速リフレッシュを選択することはできるが常に1だけが読み出される

bit1　CPUモード†
（1→V30／0→80286/80386）

bit0　不明‡
起動状態では1［DA2・RA21・VX21調べ］
DA2・RA21のITF ROMで参照あり

I/Oポート　意　味

[WRITE]　CPUシャットダウン

ビット　意　味

bit7〜3　未使用‡

bit2　リフレッシュモード選択‡
1→通常リフレッシュ（V30選択時）／0→高速リフレッシュ（80286/80386選択時）

bit1　不明‡
（1→V30選択時／
0→80286/386選択時）

bit0　CPUモード選択†
（1→V30選択／0→80286/80386選択）
出力する値によって80286(80386)とV30の切り換えが行える。そのときCPU(NDP)のリセット端子がアクティブに、A20はマスク状態になる

■ CPU A20マスク解除［286以上の機種］

I/Oポート　意　味

00F2H　[READ]　DA2調べ。RA21・H98SにはREADはなし‡

ビット　意　味

bit7〜1　――

bit0　A20状態（1→A20マスク状態／0→A20マスク解除状態）

[WRITE]　CPU A20マスク解除
任意の値を出力するとCPUのアドレスバスA20のマスクが解除される。A20をマスクするには00F6Hか00F0Hに出力する

■ CPU A20制御［80286以上搭載機］

I/Oポート　意　味

00F6H　[READ]　不明（RA21は読み出し不可能、DA2は読み出し可能）‡

[WRITE]　CPU A20制御

ビット　意　味

bit7〜2　未使用のようだが0で使用するほうがよいはず

bit1〜0　CPUのアドレスバスA20のマスク制御
（11B→A20マスク／
10B→A20マスク解除）
OUT F6H、02HとOUT F2H、*xx*Hの動作は同じ

『テクニカルデータブック』に記載されている方法では制御できない

■ DMAアクセス制御等［80286以上搭載機］‡

I/Oポート　意　味

0439H　[READ/WRITE]

ビット　意　味

bit7　未使用‡

bit6　未使用‡

bit5　不明

bit4　不明

bit3　未使用‡

bit2　DMA A23-20制御†
（1→1Mバイト以上のアドレスへのDMAアクセス禁止／0→1Mバイト以上のアドレスへのDMAアクセス許可）

bit1　不明

bit0　未使用‡

変更する場合、他のビットは保存しておく。

■ BIOS/ITF バンク切り換え［H98S・NS・DA2・RA21・VX 調べ］†

I/Oポート　意　味

043DH　［READ］ H98S 調べ。NS・DA2・RA21・VX ではなし。

［WRITE］ BIOS/ITF バンク切り換え

00011000B(18H)	H98S で ITF ROM
00010010B(12H)	BIOS 選択
00010000B(10H)	DA2・RA21・RL(NORMAL) で ITF ROM、H98S で SETUP-MENU ROM
00000010B(02H)	H98S・NS で OUT している
00000000B(00H)	RL(HIRESO) で ITF ROM

ITF ROM を選択すると、F8000～FFFFFH は ITF ROM に切り替わる。ITF ROM に切り換えた場合、E8000～F7FFFH の状態は、RA21 では BIOS ROM/RAM のまま、VX21 では FFH。リセット時は ITF ROM が選択される。ディスクブート時には BIOS が選択されている。

■ 各種バンク切り換え［DA2・RA21 調べ］†

I/Oポート　意　味

043FH　［WRITE］ 各種バンク切り換え

11000n00B　内蔵 55 ハードディスク BIOS RAM/ROM 選択
(n=1(C4H) → RAM アクティブ／n=0(C0H) → ROM アクティブ)

110000n0B　内蔵 27 ハードディスク BIOS RAM/ROM 選択
(n=1(C2H) → RAM アクティブ／n=0(C0H) → ROM アクティブ)
ポート 053DH で内蔵 27 ハードディスク BIOS をイネーブルにしたとき有効。ただし拡張スロットに D7000～D7FFFH の範囲のメモリが存在するときは内蔵 ROM をアクティブにしてはならない。

100000n0B　8-9 バンク制御
(n=0(80H) → 内部 RAM 選択／n=1(82H) →拡張スロット選択)
I/O バンク方式メモリ・ボードのため、8-9 バンクから拡張スロット上のメモリへのアクセスの可否を制御する。
「内部 RAM」を選択すると、8-9 バンクから拡張スロット上のメモリにはアクセスできなくなる。
「拡張スロット」を選択すると、RAM ウィンドウに 00000～9FFFFH の範囲内の領域がマッピングされている場合、RAM の存在しない領域がマッピングされているか、BIOS RAM をマッピングしていて BIOS RAM アクセス・ディスエーブルになっているときに、内部 RAM が切り離され、拡張スロット上のメモリにアクセスできる。
リセット時は「内部 RAM 選択」状態になっている。
386 CPU 動作時に「内部 RAM」が選択されているときは、拡張スロット上の 80000～9FFFFH のメモリにアクセスが起こることはない。
V30 CPU 動作時に「内部 RAM」が選択されているときは、拡張スロット内のマイクロスイッチが押されていなければ拡張スロットの 80000～9FFFFH のメモリにはアクセスが起こらないが、マイクロスイッチが押されていると、CPU が 8-9 バンクにアクセスしたとき拡張スロット上のメモリにもアクセスが起こるので注意しなければならない。
以上は RA21 における調査結果。

I/Oポート　意　味

RA2 ではつぎの OUT 0461H 実行まで制御が反映されない。RAM ウィンドウにどの領域が割り当てられていても「拡張スロット」を選択すると拡張スロット上のメモリにアクセスする。

010000n0B　不明‡ (n=1(42H) → 386-20MHz［RA21 調べ］
n=0(40H) → 386-16MHz［RA21 調べ］)
RA21 の ITF ROM では、386 CPU 動作時、クロック周波数 16MHz のとき n=0 で、20MHz のとき n=1 で出力される。

001000n0B　VRAM/EMS バンク切り換え
(n=1(22H) → EMS／n=0(20H) → VRAM)
NEC 方式 EMS（ページフレームが B0000H から始まり、VRAM とのバンク切り換えになっている）で、ページフレームと VRAM を切り換えるときに使用される。リセット時は VRAM が選択される。

■ RAM ウィンドウ・マッピング設定［DA2・RA21 調べ］†

I/Oポート　意　味

0461H　［WRITE］ RAM ウィンドウ・マッピング設定
RAM ウィンドウ（8, 9 バンク）にマッピングするメモリのアドレスを指定する。80000H 番地以降の内部 RAM を 20000H 番地 (128K バイト) 単位で指定できる。アドレスの上位 8 ビット (bit23～bit17) を本ポートの bit7～1 に出力する。bit0 の値は無視される。存在しないメモリをマッピングした場合の動作については「8-9 バンク制御」の項を参照。
V30 CPU 動作時も 1M バイト以上のメモリをマッピングしてアクセスできる。
RAM ウィンドウ経由で 1M バイト以上のメモリにアクセスするときは、A20 のマスク状態に影響されない。
ハイレゾ・モードの RAM ウィンドウとは異なり、RAM ウィンドウの位置にも固有の物理メモリが存在する。
RAM ウィンドウに 80000～9FFFFH 以外の内部 RAM をマッピングした場合、80000～9FFFFH をマッピングしたときと比べて 15%程度（RA21-20MHz にて）アクセス速度が遅くなる。
リセット時は 80000～9FFFFH が選択されている。

■ メモリ・イネーブル制御［DA2・RA21 調べ］†

I/Oポート　意　味

053DH　［WRITE］ メモリ・イネーブル制御

ビット	意　味
bit7	サウンド BIOS［DA・DS］(1 →イネーブル／0 →ディスエーブル)
bit6	27 タイプ内蔵ハードディスクドライブ用 HD-BIOS イネーブル制御 (1 →イネーブル／0 →ディスエーブル) D7000～D7FFFH に、内蔵している HD-BIOS を出現させるかどうかを制御する。この ROM のアクセス速度は拡張スロットに実装された ROM と同じ。ROM がイネーブルになっている場合も拡張スロットに MRC/MWC/A19 信号は出力される。リセット時はディスエーブル。
bit5	未使用‡
bit4	未使用‡
bit3	未使用‡
bit2	BIOS RAM アクセス・イネーブル制御 (1 →ディスエーブル／0 →イネーブル) RA2 では E0000～FFFFFH に対して、RA21 では C0000～FFFFFH に対して機能する。そのアドレスを RAM ウィンドウにマッピングしたとき、アクセスを許可するかどうかを制御する。

I/Oポート	意　味
	アクセス・イネーブル状態でなければ前記アドレスに対して RAM ウィンドウから読み書きすることはできない。 リセット時はアクセス・イネーブル。起動状態ではディスエーブル。
	bit1　BIOS RAM/ROM SELECT (1→BIOS RAM 選択／0→BIOS ROM 選択) E8000～FFFFFH の BIOS 領域に RAM と ROM のどちらを出現させるかを選択する。 リセット時は BIOS ROM 選択。BIOS RAM に BIOS ROM の内容を転送後は BIOS RAM が選択される。
	bit0　未使用‡
	このポートは読み出しができないので、どれか1箇所のビットだけを変更したい場合でも他のビットの状態を考慮して書き込みを行わなければならない。

■ ソフトウェア DIP SW 1［DA2 調べ］†

I/Oポート	意　味
841EH	[READ] (1→OFF／0→ON) ビット　意　味 bit7　SW1-8 bit6　SW1-7 bit5　SW1-6 bit4　SW1-5 bit3　SW1-4 bit2　SW1-3 bit1　常に 1 bit0　パリティ合わせ (Odd)
	[WRITE] (1→OFF／0→ON) ビット　意　味 bit7　SW1-8 bit6　SW1-7 bit5　SW1-6 bit4　SW1-5 bit3　SW1-4 bit2　SW1-3 bit1　サウンド BIOS 使用状態 (1→使用／0→切り離し) bit0　パリティ合わせ (Odd)

■ ソフトウェア DIP SW 2［DA2 調べ］†

I/Oポート	意　味
851EH	[READ]　サウンド BIOS 使用状態取得 ビット　意　味 bit7～2　―― bit1　サウンド BIOS 使用状態 (1→使用／0→切り離し) bit0　――
	[WRITE] (1→OFF／0→ON) ビット　意　味 bit7　SW2-8 bit6　SW2-7 bit5　SW2-6 bit4　パリティ合わせ (Odd) bit3　SW2-4 bit2　SW2-3 bit1　SW2-2 bit0　SW2-1

■ ソフトウェア DIP SW 3［DA2 調べ］†

I/Oポート	意　味
861EH	[READ] ビット　意　味 bit7　パリティ合わせ (Odd) bit6　SW3-3 bit5　SW3-6 bit4　SW3-5 bit3　―― bit2　―― bit1　SW3-2 bit0　SW3-1
	[WRITE] ビット　意　味 bit7　パリティ合わせ (Odd) bit6　SW3-3 bit5　SW3-6 bit4　SW3-5 bit3　―― bit2　―― bit1　SW3-2 bit0　SW3-1

■ NESA-FO レジスタ［H98］

n	8～F	スロット番号に対応
m	D	本体内蔵システム専用スロット
	E	拡張 BOX 内オプションスロット

I/Oポート	意　味
8*nm*0H	[READ]　レジスタ0リード（機能 ID1）[E²PROM、PROM、フロッピーディスク媒体] ボードメーカー・コード1
	[WRITE]　なし [E²PROM、PROM、フロッピーディスク媒体]
	[READ]　なし [PAL]
	[WRITE]　レジスタ0ライト（機能 ID1）[PAL]
8*nm*1H	[READ]　レジスタ1リード（データロードレジスタ）[E²PROM] ビット　意　味 bit7　EDO（E²PROM からのデータ出力） bit6　ECL（E²PROM のクロック入力） bit5　ECS（チップセレクト） bit4　EDI（データ入力） bit3～0　0
	[WRITE]　レジスタ1ライト（データロードレジスタ）[E²PROM] ビット　意　味 bit7　0 bit6　ECL（E²PROM のクロック入力） bit5　ECS（チップセレクト） bit4　EDI（データ入力） bit3～0　0
	[READ]　レジスタ1リード（データロードレジスタ）[PROM] ROM、PAL リードデータ
	[WRITE]　レジスタ1ライト（データロードレジスタ）[PROM] ビット　意　味 bit7　ACC（NESA-FO のアドレスカウンタリセット） (1→カウンタにリセットパルスが出る) bit6～0　0
	[READ]　レジスタ1リード（データロードレジスタ）[フロッピーディスク媒体]
	[WRITE]　レジスタ1ライト（データロードレジスタ）[フロッピーディスク媒体]
	[READ]　レジスタ1リード（セットアップデータリードバイト1）[PAL]
	[WRITE]　なし [PAL]
8*nm*2H	[READ]　レジスタ2リード（機能 ID2）[E²PROM、PROM、フロッピーディスク媒体] ボードメーカー・コード2 ボードメーカー・コード1、2は、ボードメーカー名（大文字アルファベット3文字）をアスキーコードにし、それを2バイトに圧縮したもの

I/Oポート	意 味
	[WRITE] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] なし [PAL] [WRITE] レジスタ2ライト（機能ID2）[PAL]
8*nm*3H	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ3ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] レジスタ3リード（セットアップデータリードバイト2）[PAL] [WRITE] なし [PAL]
8*nm*4H	[READ] レジスタ4リード（機能ID3）[E^{2}PROM、PROM、フロッピーディスク媒体] 製品コード（ボードに一対一で対応する製品コード） [WRITE] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] なし [PAL] [WRITE] レジスタ4ライト（機能ID2）[PAL]
8*nm*5H	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ5ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] レジスタ5リード（セットアップデータリードバイト3）[PAL] [WRITE] なし [PAL]
8*nm*6H	[READ] レジスタ6リード（機能ID4）[E^{2}PROM、PROM、フロッピーディスク媒体] ビット 意 味 bit7～4 コンプ・コード（ボードの改版による版数） bit3～2 ボード区分 (11B→未実装／10B→リザーブ／01B→新ボード／00B→従来ボード) bit1～0 セットアップ情報タイプ (11B→E^{2}PROM／10B→PROM／01B→PAL／00B→フロッピーディスク媒体) [WRITE] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] なし [PAL] [WRITE] レジスタ6ライト（機能ID4）[PAL]
8*nm*7H	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ7ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] レジスタ7リード（セットアップデータリードバイト4）[PAL] [WRITE] なし [PAL]
8*nm*8H	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ8ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] なし [PAL] [WRITE] レジスタ8ライト（ボードコントロール）[PAL]
8*nm*9H	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ9ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] レジスタ9リード（セットアップデータリードバイト5）[PAL] [WRITE] なし [PAL]
8*nm*AH	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ10ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] なし [PAL] [WRITE] レジスタ10ライト（DMA、割り込み、ウェイト数）[PAL]

I/Oポート	意 味
8*nm*BH	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ11ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] レジスタ11リード（セットアップデータリードバイト6）[PAL] [WRITE] なし [PAL]
8*nm*CH	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ12ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] なし [PAL] [WRITE] なし [PAL]
8*nm*DH	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ13ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] レジスタ13リード（セットアップデータリードバイト7）[PAL] [WRITE] なし [PAL]
8*nm*EH	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ14ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] なし [PAL] [WRITE] なし [PAL]
8*nm*FH	[READ] なし [E^{2}PROM、PROM、フロッピーディスク媒体] [WRITE] レジスタ15ライト [E^{2}PROM、PROM、フロッピーディスク媒体] [READ] レジスタ15リード（セットアップデータリードバイト8）[PAL] [WRITE] なし [PAL]

■ 電源制御等［H98］†

I/Oポート	意 味
0431H	[READ] ‡ [WRITE] 10001000B(88H)→電源OFF 10000011B(83H)→電源スイッチのロック

■ CPUキャッシュ制御［H98S調べ］†

I/Oポート	意 味
9896H	[READ] ‡ [WRITE] 00000000B(00H)→キャッシュON 00000100B(04H)→キャッシュOFF

■ NOTE RAMバンク（DA000～DBFFFH）切り換え†

I/Oポート	意 味
0E8EH	[WRITE] RAMバンク設定［98N調べ］ BF～04H→RAM DRIVE 03～00H→NOTE MENU [WRITE] RAMバンク設定［NS調べ］ FFH→MAINメモリ00000～01FFFHバックアップなし FEH→MAINメモリ02000～03FFFH : F0H→MAINメモリ1E000～1FFFFH EF～E0H→MAINメモリ20000～3FFFFH DF～D0H→MAINメモリ40000～5FFFFH CF～C0H→MAINメモ60000～7FFFFH

I/Oポート	意　味
	BF〜B0H → MAIN メモリ 80000〜9FFFFH AF〜A8H →裏 RAMB0000〜BFFFFH A7〜A6H →予約（未使用）バックアップあり A5〜A2H → NOTE MENU A1H → NOTE システム値 A0H →フォーマット形式‡ 9F〜00H → NOTE DISK

ポート 1E8EH＝81H のときのみ

■ NOTE バンクフラグ（0E8EH のバンク切り換え）†

I/Oポート	意　味
1E8EH	[WRITE] バンクフラグ設定 1000000*n*B　RAM バンクフラグ （*n*=1(81H)→RAM イメージ／*n*=0(80H)→ROM イメージ）

■ ROM バンク（D8000〜D9FFFH）切り換え［NS］†

I/Oポート	意　味
4E8EH	[READ] ROM バンク状態取得 80H ←バンク 0（NOTE 用 ROM） : 83H ←バンク 3 [WRITE] ROM バンク状態設定 80H ←バンク 0（NOTE 用 ROM） : 83H ←バンク 3 バンク 4〜は、E8000H 番地以降の ROM イメージ。通常はバンク 0 を選択。

■ NOTE 用 LED コントロール［NS・NV・NS/E］†

I/Oポート	意　味
5E8EH	[READ/WRITE]　LED 状態取得、設定 ビット　意　味 bit7〜4　—— bit3　FDD-LED（1→アンバー点灯／0→消灯） bit2　—— bit1　RAM-LED（1→アンバー点灯／0→消灯） bit0　RAM-LED（1→赤色点灯／0→消灯）

■ NOTE 用内蔵モデムコントロール‡

I/Oポート	意　味
7E8EH	[WRITE] モデム制御 ビット　意　味 bit7　モデムの電源制御（1→ON／0→OFF） bit6〜0　不明

■ NOTE 用コントロール［NS・NV・NS/E］†

I/Oポート	意　味
AE8EH	[READ/WRITE]　各種状態取得、設定 ビット　意　味 bit7　パリティ合わせ (Odd) bit6 bit5〜4　BEEP 音量調節 11B →最大 : 00B →消音 bit3　LCD 階調（1→2 色／0→8 色）

I/Oポート	意　味
	bit2　LCD 反転（1→背景が青、黒／0→背景が白） bit1〜0　GRPHmode

■ NOTE 用 RAM モード［NS・NS/E］†

I/Oポート	意　味
BE8EH	[READ/WRITE]　RAM モード取得、設定 ビット　意　味 bit7　パリティ合わせ (Odd) bit6〜1　—— bit0　RAMmode（1→拡張 RAM／0→互換ドライブ）

■ NOTE 用 LCD コントロール［NS］†

I/Oポート	意　味
CE8EH	[READ/WRITE]　LCD 状態取得、設定 ビット　意　味 bit7　LCD ライト（1→ON／0→OFF） bit6〜0　——

■ NOTE 用コントロール［NV・NS/E］†

I/Oポート	意　味
08F2H	[READ]　LCD 状態取得 ビット　意　味 bit7　LCD ライト（1→ON／0→OFF） bit6〜0　——

■ AC アダプタステータス［NOTE］†

I/Oポート	意　味
8810H	[READ]　(NS 調べ) ビット　意　味 bit7〜3　—— bit2　バッテリー稼動状態（1→AC アダプター使用中／0→バッテリー駆動中） bit1　バッテリー容量（1→LOW バッテリー／0→充電できている） bit0　——

■ 動作 CPU［DA・NS 調べ］‡

I/Oポート	意　味
9892H	[READ/WRITE]（ライトしても動作 CPU は変わらない‡） ビット　意　味 bit7〜4　—— bit3　CPU（1→6MHz［NS］、V30［DA］／0→12MHz［NS］、386［DA］） bit2〜0　——

■ 動作クロック［DA・NS 調べ］‡

I/Oポート	意　味
9894H	[READ/WRITE]（ライトしても動作クロックは変わらない‡） ビット　意　味 bit7〜4　—— bit3　CLOCK（1→6MHz［NS］、V30［DA］／0→12MHz［NS］、386［DA］） bit2〜1　—— bit0　0

割り込み一覧

PC-9801でサポートしているBIOSの一覧である。各ファンクションの詳細は、『テクニカルデータブック』に譲る。ただし、一般に公開されていないファンクションや独自に解析したファンクションの解説を、一覧の後ろにつけた。

INT 18H

■ キーボード BIOS

番号	意味
00H	キーデータの読み出し［ノーマル・ハイレゾ・LT・HA］
01H	キーバッファ状態のセンス［ノーマル・ハイレゾ・LT・HA］
02H	シフトキー状態のセンス［ノーマル・ハイレゾ・LT・HA］
03H	キーボードインターフェイスの初期化［ノーマル・ハイレゾ・LT・HA］
04H	キー押下状態のセンス［ノーマル・ハイレゾ・LT・HA］
05H	バッファからのキーコード読み出し［ノーマル・ハイレゾ・LT・HA］
06H	バッファの初期化［ハイレゾ］
07H	シフト状態とキーデータの読み出し［ハイレゾ］
08H	シフト状態とキーデータのセンス［ハイレゾ］
09H	キーデータの作成［ハイレゾ］

■ CRT BIOS

番号	意味
0AH	CRTのモード設定［ノーマル・ハイレゾ・LT・HA］
0BH	CRTモードのセンス［ノーマル・ハイレゾ・LT・HA］
0CH	テキスト画面の表示開始［ノーマル・ハイレゾ・LT・HA］
0DH	テキスト画面の表示停止［ノーマル・ハイレゾ・LT・HA］
0EH	1つの表示領域の設定［ノーマル・ハイレゾ・LT・HA］
0FH	複数の表示領域の設定［ノーマル・ハイレゾ・LT・HA］
10H	カーソルタイプの設定［ノーマル・ハイレゾ・LT・HA］
11H	カーソルの表示開始［ノーマル・ハイレゾ・LT・HA］
12H	カーソルの表示停止［ノーマル・ハイレゾ・LT・HA］
13H	カーソル位置の設定［ノーマル・ハイレゾ・LT・HA］
14H	フォントパターンの読み出し（16ドット）［ノーマル・LT・HA］
15H	押されているライトペンの位置の読み出し［ノーマル］
16H	テキストVRAMの初期化［ノーマル・ハイレゾ・LT・HA］
17H	ブザーの起呼［ノーマル・ハイレゾ］
18H	ブザーの停止［ノーマル・ハイレゾ］
19H	ライトペン押下状態の初期化［ノーマル］
1AH	ユーザ文字の定義（16×16ドット）［ノーマル・LT・HA］
1BH	漢字キャラクタジェネレータアクセスモードの設定［ノーマル・ハイレゾ］
1CH	CRTの初期化［ハイレゾ］
1DH	表示幅の設定［ハイレゾ］
1EH	カーソルタイプの設定［ハイレゾ］
1FH	フォントパターン（24ドット）の読み出し［ハイレゾ］
20H	ユーザ作成文字（24ドット）の定義［ハイレゾ］
21H	メモリスイッチの読み出し［ハイレゾ・LT・HA］
22H	メモリスイッチの書き込み［ハイレゾ・LT・HA］
23H	ブザー周波数の設定［ハイレゾ］
24H	ブザーの時間設定鳴動［ハイレゾ］
25H	エミュレーションVRAMの確保/アドレスの取得［LT・HA］
26H	エミュレーションVRAM上のテキストの表示［LT・HA］
27H	領域のスクロール［LT・HA］
28H	一文字の表示［LT・HA］
29H	テキストスクロール［H98］
2AH	拡張アトリビュートのモード設定［H98］
2BH	拡張アトリビュートモードの取得［H98］
2CH	罫線色の設定［H98］
2DH	未公開（不明）［H98S］

■ グラフィック BIOS［ノーマル］

番号	意味
40H	グラフィック画面の表示開始［ノーマル］
41H	グラフィック画面の表示停止［ノーマル］
42H	表示領域の設定［ノーマル］
43H	パレットレジスタの設定［ノーマル］
44H	ボーダーカラーの設定［ノーマル］
45H	ドットの書き込み［ノーマル］
46H	ドットの読み出し［ノーマル］
47H	直線、矩形の描画［ノーマル］
48H	円弧の描画［ノーマル］
49H	グラフィック文字の描画［ノーマル］
4AH	描画モードの設定［ノーマル］

INT 19H

■ RS-232C BIOS

番号	意味
00H	初期化（互換モード）［ノーマル・ハイレゾ・LT・HA］
01H	フロー制御を伴う初期化（互換モード）［ノーマル・ハイレゾ・LT・HA］
02H	受信データ長の取得［ノーマル・ハイレゾ・LT・HA］
03H	データの送信［ノーマル・ハイレゾ・LT・HA］
04H	データの受信［ノーマル・ハイレゾ・LT・HA］
05H	8251Aへのコマンド出力［ノーマル・ハイレゾ・LT・HA］
06H	ステータス情報の取得［ノーマル・ハイレゾ・LT・HA］
07H	初期化（拡張モード）［ハイレゾ］

INT 1AH

■ CMT（カセットテープ）BIOS

番号	意味
01H	カセットのモータOFF［ノーマル］
02H	カセットのモータON（データリードモード）［ノーマル］
03H	カセットのモータON（データライトモード）［ノーマル］
04H	データライト［ノーマル］
05H	データリード［ノーマル］

■ プリンタ BIOS

番号	意味
10H	初期化［ノーマル・ハイレゾ・LT・HA］
11H	データの出力（ウェイトモード）［ノーマル・ハイレゾ・LT・HA］
12H	ステータスの取得［ノーマル・ハイレゾ・LT・HA］
14H	データの出力（ノーウェイトモード）［ハイレゾ］
15H	データの出力（チェックレスモード）［ハイレゾ］
16H	初期化（ウェイト時間設定）［ハイレゾ］
17H	プリンタインターフェイスフルセントロニクス化［H98ノーマル］
18H	ステータス取得［H98ノーマル］
19H	プリンタモード取得［H98ノーマル］
1AH	プリンタインターフェイス簡易セントロニクス化［H98ノーマル］
30H	複数バイトデータ出力［ノーマル・ハイレゾ・LT・HA］

INT 1BH

■ ディスク BIOS

番号	意味
10H	ヘッドの移動（SEEK）［両用・1MB・640KB］
F0H	モータOFF（未公開）［IDE］

番号	意味
*n*1H	ベリファイ（VERIFY）［両用・1MB・640KB・320KB・SASI・SCSI］
*n*2H	診断のための読み出し（READ DIAGNOSTIC）［両用・1MB］
*n*2H	データの書き込みとベリファイ（WRITE & VERIFY）［SCSI］
03H	初期化（INITIALIZE）［両用・1MB・640KB・SASI・SCSI］
03H	初期化（INITIALIZE）［320KB］
83H	モータ停止モードの設定［両用1MB I/F MODE］
83H	新イニシャライズ［両用640KB I/F MODE］
04H	センス（SENSE）［両用・1MB・640KB・320KB］
04H	センス（SENSE）［SASI］
04H	センス（不明）［SCSI］
44H	センス2（SENSE2）［SCSI］
84H	新センス［両用］
84H	新センス［SASI］
84H	新センス［SCSI］
*n*5H	データの書き込み（WRITE DATA）［両用・1MB・640KB・320KB・SASI・SCSI］
*n*6H	データの読み出し（READ DATA）［両用・1MB・640KB・320KB・SASI・SCSI］
*n*7H	シリンダ0へシークする（RECALIBRATE）［両用・1MB・640KB・SASI・SCSI］
08H	代替トラックの指定（ASSIGN ALTERNATE TRACK）［SASI］
*n*9H	デリーテッドデータの書き込み（WRITE DELETED DATA）［1MB・640KB・両用］
09H	トラック/セクタの代替処理（REASSIGN BLOCKS）［SCSI・ESDI］
*n*AH	IDの読み出し（READ ID）［両用・1MB・640KB］
0AH	セクタ長の指定（SET SECTOR）［SCSI・ESCI］
0BH	不良トラックのフォーマット（FORMAT BAD TRACK）［SASI］
*n*CH	デリーテッドデータの読み出し（READ DELETED DATA）［1MB］
0CH	不良セクタリストの取得（READ DEFECT DATA）［SCSI・ESDI］
8CH	不良セクタ数の取得（READ DEFECT NUMBER）［SCSI・ESDI］
*n*DH	トラックのフォーマット（FORMAT TRACK）［両用・1MB・640KB］
*n*DH	トラックのフォーマット（FORMAT DRIVE）［320KB］
*n*DH	フォーマット（FORMAT TRACK/DRIVE）［SASI］
*n*DH	フォーマット（FORMAT DRIVE/TRACK）［SCSI］
*n*EH	SET OPERATION MODE［両用1MB I/F MODE］
0EH	SET OPERATION MODE［320KB］
*n*FH	リトラクト（RETRACT）［SASI・SCSI］

■ SCSIインターフェイス共通BIOS

番号	意味
00H	RESET
03H	NEGATE ACK
07H	SELECT-WITHOUT-ATN
09H	SELECT-WITHOUT-ATN & TRANSFER
18H	TRANSFER DATA OUT
19H	TRANSFER DATA IN
1AH	TRANSFER COMMAND
1BH	TRANSFER STATUS
1CH	TRANSFER UNSPECIFIED INFO. OUT
1DH	TRANSFER UNSPECIFIED INFO. IN
1EH	TRANSFER MESSAGE OUT
1FH	TRANSFER MESSAGE IN

INT 1CH

■ カレンダ/インターバルタイマBIOS

番号	意味
00H	日付と時刻の読み出し［ノーマル・ハイレゾ・LT・HA］
01H	日付と時刻の設定［ノーマル・ハイレゾ・LT・HA］
02H	インターバルタイマの設定（シングルイベント）［ノーマル・ハイレゾ・LT・HA］
03H	（内部使用）［ノーマル・LT・HA］
03H	タイマキャンセル［ハイレゾ］
04H	インターバルタイマの設定（ワンショットモード）［ハイレゾ］
05H	インターバルタイマの設定（リピートモード）［ハイレゾ］
06H	ビープ機能［ハイレゾ］
07H	アラーム機能の設定［HA］
08H	アラーム設定の解除［HA］
09H	アラーム設定状態の読み出し［HA］

INT 1DH

■ グラフィックBIOS［ハイレゾ・LT・HA］

番号	意味
00H	グラフィックBIOSの初期化（GINIT）［ハイレゾ・LT・HA］
01H	モード設定（GSCREEN）［ハイレゾ］
01H	ビューポートの初期化（GSCREEN）［LT・HA］
02H	描画領域の指定（GVIEW）［ハイレゾ・LT・HA］
03H	フォアグラウンドカラー、バックグラウンドカラーの指定（GCOLOR1）［ハイレゾ・LT・HA］
04H	パレット番号と表示色の対応（GCOLOR2）［ハイレゾ］
05H	描画領域のクリア（GCLS）［ハイレゾ・LT・HA］
06H	ドットの書き込み（GPSET）［ハイレゾ・LT・HA］
07H	直線、矩形の描画（GLINE）［ハイレゾ］
07H	直線、矩形の描画（GLINE）［LT・HA］
08H	円、楕円の描画（GCIRCLE）［ハイレゾ・LT・HA］
09H	指定色による塗りつぶし（GPAINT1）［ハイレゾ・LT・HA］
0AH	タイルパターンによる塗りつぶし（GPAINT2）［ハイレゾ・LT・HA］
0BH	画面イメージの格納（GGET）［ハイレゾ・LT・HA］
0CH	画面イメージの復帰（GPUT1）［ハイレゾ・LT・HA］
0DH	日本語の描画（GPUT2）［ハイレゾ・LT・HA］
0EH	画面イメージの移動（GROLL）［ハイレゾ・LT・HA］
0FH	指定ドットのパレット番号の取得（GPOINT2）［ハイレゾ・LT・HA］
10H	画面イメージの展開（GCOPY）［LT］
10H	表示領域の設定（GSCROLL）［ハイレゾ］
11H	グラフィック画面の表示開始（GSTART）［ハイレゾ］
12H	グラフィック画面の表示停止（GSTOP）［ハイレゾ］
13H	グラフィックBOISの終了（GTERM）［ハイレゾ］

INT 1FH

■ マルチタスク対応割り込み［ハイレゾ］

番号	意味
80H	キーボードBIOSアイドル割り込み［ハイレゾ］
81H	キーボード割り込みハンドラ内で使用［ハイレゾ］
82H	プリンタBUSY時の割り込み［ハイレゾ］
83H	RS-232C拡張モード時の受信割り込み発生通知［ハイレゾ］
88H	キーボード割り込みハンドラ内で使用［ハイレゾ］

■ HAアラームBIOS

番号	意味
8CH	INT 0AH割り込みハンドラ内で使用（未公開）［HA］
8EH	アラーム通知［HA］
8FH	アラーム鳴動用タイマ［HA］

■ プロテクトメモリ BIOS

番号	意味
90H	プロテクトメモリとのメモリ転送［80286・80386・80486搭載機］
91H	プロテクトメモリ上でのプログラム実行［80286・80386・80486 搭載機］

■ 電源 BIOS

番号	意味
98H	電源 OFF［LT・HA・NOTE・H98］

■ ROM 辞書アクセス［LT・HA］

番号	意味
A0H	初期化［LT・HA］
A1H	見出しのサーチ［LT・HA］
A2H	漢字表記の読み出し［LT・HA］
A3H	システム標準辞書、ユーザ単語辞書の学習［LT・HA］
A4H	学習ページのサーチ／読み出し［LT・HA］
A6H	ユーザ単語辞書への登録／学習ページへの単語登録［LT・HA］
A7H	ユーザ単語辞書内の削除／学習ページ内の削除［LT・HA］

■ メモリカード BIOS［HA］

番号	意味
B4H	データの読み出し［HA］
B5H	データの書き込み［HA］
B6H	カードサイズ取得［HA］

■ システム BIOS［HA］

番号	意味
B9H	スタンバイモードへの移行［HA］

■ NESA–FO BIOS［H98］

番号	意味
C0H	対象スロット番号の取得（1）［H98］
C1H	対象スロット番号の取得（2）［H98］
C2H	機能 ID の取得［H98］
C3H	NESA–FO レジスタ設定値の取得［H98］
C4H	NESA–FO 割り付け設定情報の取得［H98］
C5H	マウス割り込みレベルの取得［H98］
C7H	NON DUCUMENT［H98］

■ DMA BIOS［H98］

番号	意味
D2H	チャネルアロケート［H98］
D3H	DMA チャネル初期設定［H98］
D5H	DMA パラメータセット［H98］
D6H	DMA パラメータリードバック［H98］
D7H	チャネルリリース［H98］

■ 拡張 ROM BIOS［H98］

番号	意味
D9H	拡張 ROM 領域割り込み制御ブロック更新［H98］

INT DCH

■ MS–DOS 拡張機能呼び出し

番号	意味
0AH	RS–232C ポートの初期化［ハイレゾ］
0CH	キーの取得［ノーマル・ハイレゾ］
0CH	キーの取得［LT］
0DH	キーの設定
0EH	RS–232C ポートの操作［ノーマル・ハイレゾ］
0EH	RS–232C ポートのデータ長通知［LT・HA］
0FH	CTRL＋ファンクションキーのソフトキー化／解除
10H	直接コンソール出力
11H	プリンタモードの変更
12H	MS–DOS 製品バージョンの取得（未公開）
13H	MS–DOS ドライブ名と DA/UA 対照リストの取得（未公開）

■ 日本語入力拡張機能ファンクション

番号	意味
E0H	アプリケーションへの開放
E1H	アプリケーションからの使用禁止
E2H	キーボードからの日本語入力の禁止／許可
E3H	学習機能の有無を設定
E4H	ローマ字列をカナ文字列に変換
E5H	1 バイト JIS 文字列を全角文字列に変換
E6H	1 バイト JIS 文字列を 2 バイト半角文字列（シフト JIS）に変換
E7H	辞書のオープン
E8H	辞書のクローズ
E9H	語句の登録
EAH	語句の削除
EBH	語句の学習（文）
ECH	語句の変換（単文節変換：最初の候補）（文）
EDH	語句への変換（単文節変換：次候補）（文）
EEH	語句の変換（単文節変換：前候補）
EFH	日本語入力モードに入る
F0H	日本語入力モードから抜ける
F1H	日本語入力モードのセット
F2H	日本語入力モードの取得
F3H	2 バイト JIS をシフト JIS に変換
F4H	シフト JIS を 2 バイト JIS に変換
F7H	AI かな漢字変換ドライバの有無の取得（逐）（連）
F8H	辞書の先読みと逐次変換（逐）（連）
F9H	連文節変換（最初の候補）
FAH	連文節変換（次候補）（逐）（連）
FBH	連文節変換（前候補）（逐）（連）
FCH	学習（連文節）（逐）（連）
FDH	先読み機能の有無の設定（連）

未公開ファンクションの解説

INT 18H

■ キーボード BIOS

番号	意味
05H	バッファからのキーコード読み出し［ノーマル・ハイレゾ・LT・HA］ キーバッファの先頭から1文字分のキー入力データを取り出し、キーコードとキーデータを返す。バッファが空のときはすぐに呼出し元に戻る。 入力　なし 出力　BH←キーデータの有無 　(01H→あり／00H→なし) 　AH←キーコード、AL←キーデータ 破壊　AX、BH テクニカルデータブックではハイレゾのみとなっているが、実際はその他のモードでも使用可能。

INT 1BH

■ ディスク BIOS

番号	意味
05H	バッファからのキーコード読み出し［ノーマル・ハイレゾ・LT・HA］ キーバッファの先頭から1文字分のキー入力データを取り出し、キーコードとキーデータを返す。バッファが空のときはすぐに呼出し元に戻る。
F0H	モータ OFF（未公開）［IDE］ NOTE 搭載ハードディスクドライブのモータを OFF にする。ハードディスクドライブにアクセスするファンクションを実行すれば自動的にモータ ON となる。ただし、READY 状態になるまでに通常4〜5秒かかる。 入力　AH←1/1/1/1/0/0/0/0 　AL←DA/UA 出力　CF←終了条件（1→異常終了／0→正常終了） 　AH←ステータス情報

INT 1FH

■ プロテクトメモリ BIOS

番号	意味
90H	プロテクトメモリとのメモリ転送［80286・80386・80486搭載機］ 入力　ES:BX←セグメントディスクリプタのポインタ 　SI←転送元アドレスのオフセット（セグメントは DS のセレクタで指定） 　DI←転送先アドレスのオフセット（セグメントは ES のセレクタで指定） 　CX←転送バイト数（0のとき64Kバイト） 　●セグメントディスクリプタの構造 　＋00〜0FH　リザーブ（ユーザ設定不要） 　＋10〜17H　DS 用（ユーザ設定必要） 　＋18〜1FH　ES 用（ユーザ設定必要） 　＋20〜27H　CS 用（ユーザ設定不要） 　＋28〜2FH　SS 用（ユーザ設定不要） 出力　CF＝1→エラー（CPU が V30 のとき） 　CF＝0→正常終了
91H	プロテクトメモリ上でのプログラム実行［80286・80386・80486 搭載機］ 入力　ES:BX←セグメントディスクリプタのポインタ 　DH←マスタ PIC イニシャライズ時の ICW2（先頭割り込み番号） 　DL←スレーブ PIC イニシャライズ時の ICW2（先頭割り込み番号） 　●セグメントディスクリプタの構造 　＋00〜07H　リザーブ（ユーザ設定不要） 　＋08〜0FH　GDT（ユーザ設定必須） 　＋10〜17H　IDT（ユーザ設定必須） 　＋18〜1FH　DS 用（ユーザ設定任意） 　＋20〜27H　ES 用（ユーザ設定任意） 　＋28〜2FH　SS 用（ユーザ設定必須） 　＋30〜37H　CS 用（ユーザ設定必須） 出力　CF＝1→エラー（CPU が V30 のとき） 　CF＝0→正常終了 備考　INT 1FH 実行前の準備 　SHUT 0←0（OUT 37H、0EH） 　PUSH CS 　PUSH（RETURN OFFSET） 　0000:0404H←SP 　0000:0406H←SS 　ユーザプログラムは IRET で終了

■ 電源 BIOS

番号	意味
98H	電源 OFF［LT・HA・NOTE・H98］ 入力　なし 出力　なし（？）

■ INT DCH MS-DOS 拡張機能呼び出し

MS-DOS の製品バージョンによってサポートしているファンクションの範囲が異なるので注意すること。
ファンクション番号は CL レジスタに設定する。

番号	意味
12H	MS-DOS 製品バージョンの取得（未公開） 入力　AX←0000H 出力　AX←製品バージョン 　DX←機種情報 ●3.1（PS98-011） 　AX＝0001H 　DX＝0000H→初代 　DX＝0001H→E・F・M 　DX＝0002H→U 　DX＝0003H→上記以外のノーマル 　DX＝0100H→XA 　DX＝0101H→XA 以外のハイレゾ ●3.3、3.3A（PS98-013、015） 　AX＝0002H 　DX＝0000H→初代 　DX＝0001H→E・F・M 　DX＝0002H→U 　DX＝0003H→上記以外のノーマルの旧キーボード 　DX＝0004H→上記以外のノーマルの新キーボード 　DX＝0100H→XA 　DX＝0101H→XA 以外のハイレゾ 　DX＝1000H→ノーマル（0000:0481H & 87H＝81） 　DX＝1100H→ハイレゾ（0000:0481H & 87H＝81） 　DX＝1001H→ノーマル（0000:0481H & 87H＝82） 　DX＝1101H→ハイレゾ（0000:0481H & 87H＝82） 　DX＝1002H→ノーマル（0000:0481H & 87H＝84） 　DX＝1102H→ハイレゾ（0000:0481H & 87H＝84） 　DX＝1002H→ノーマル・ハイレゾ（0000:0481H & 87H != 81、82、84） 　*0000:0481H bit7 は、当初 NESA 識別用に割り振られたらしい 　*キーボード識別フラグは 0000:0481H bit6 ●3.3B（PS98-017） 　AX＝0002H 　DX＝0000H→初代 　DX＝0001H→E・F・M 　DX＝0002H→U

番号	意味
	DX＝0003H →上記以外のノーマル (0000:0458H & 40H ＝0) DX＝0004H →上記以外のノーマル (0000:0458H & 40H != 0) DX＝0100H → XA DX＝0101H → XA 以外のハイレゾ DX＝1004H → NESA ノーマル DX＝1101H → NESA ハイレゾ *0000:0458H bit6 は、NESA 関連情報のバイト。 実は 0000:0481H bit6（新キーボードフラグ）の誤りでは？ • 3.3C、3.3D（PS98–019、1002） AX＝0003H → MS–DOS 3.3C AX＝0004H → MS–DOS 3.3D DX＝0000H →初代 DX＝0001H → E・F・M DX＝0002H → U DX＝0003H →上記以外のノーマル (0000:0458H & 40H ＝00H) DX＝0004H →上記以外のノーマル (0000:0458H & 40H != 00H) DX＝0100H → XA DX＝0101H → XA 以外のハイレゾ DX＝1004H → NESA ノーマル (0000:0487H != 04H) DX＝1005H → NESA ノーマル (0000:0487H ＝04H) DX＝1101H → NESA ハイレゾ (0000:0487H != 04H) DX＝1102H → NESA ハイレゾ (0000:0487H ＝04H) *0000:0458H bit6 は、NESA 関連情報のバイト。 実は 0000:0481H bit6（新キーボードフラグ）の誤りでは？ *0000:0487H の意味は不明。 • 5.0 （PS98–1003） AX＝0101H DX＝0000H →初代 DX＝0001H → E・F・M DX＝0002H → U DX＝0003H →上記以外のノーマルの旧キーボード DX＝0004H →上記以外のノーマルの新キーボード DX＝0100H → XA DX＝0101H → XA 以外のハイレゾ DX＝1004H → NESA ノーマル (0000:0487H != 04H) DX＝1005H → NESA ノーマル (0000:0487H ＝04H) DX＝1101H → NESA ハイレゾ (0000:0487H != 04H) DX＝1102H → NESA ハイレゾ (0000:0487H ＝04H) *0000:0487H の意味は不明。 • EPSON 3.1　　不変 • EPSON 3.1R2.1　0001 • EPSON 4.01R10　0002
13H	MS–DOS ドライブ名–DA/UA 対照リストの取得（未公開） CL ← 12H、AX ← 0000H で INT DCH を実行したとき、AX > 1 ならこのファンクションが利用できる。また、PS98–011 以降の FORMAT.EXE 等はこの方法で DA/UA を取得している。 入力　DS:DX ←リスト出力バッファ（60H バイト） 出力　リスト出力バッファ • リスト出力バッファの内容 リスト出力バッファの先頭からのオフセットと対応するドライブ名を示す。英字がドライブ名。数値が対応しているところはその数値が入っている。? は不定。

+00H → A	+20H → 00H	+40H → 00H
+01H → B	+21H → D	+41H → T
+02H → C	+22H → 00H	+42H → 00H
+03H → D	+23H → E	+43H → U
+04H → E	+24H → 00H	+44H → 00H
+05H → F	+25H → F	+45H → V
+06H → G	+26H → 00H	+46H → 00H
+07H → H	+27H → G	+47 → W
+08H → I	+28H → 00H	+48H → 00H
+09H → J	+29H → H	+49 → X
+0AH → K	+2AH → 00H	+4AH → 00H
+0BH → L	+2BH → I	+4BH → Y
+0CH → M	+2CH → 00H	+4CH → 00H
+0DH → N	+2DH → J	+4DH → Z
+0EH → O	+2EH → 00H	+4EH → 00H
+0FH → P	+2FH → K	+4FH → 00H
+10H → ?	+30H → 00H	+50H → ?
+11H → ?	+31H → L	+51H → ?
+12H → ?	+32H → 00H	+52H → ?
+13H → ?	+33H → M	+53H → ?
+14H → ?	+34H → 00H	+54H → ?
+15H → ?	+35H → N	+55H → ?
+16H → ?	+36H → 00H	+56H → ?
+17H → ?	+37H → O	+57H → ?
+18H → ?	+38H → 00H	+58H → ?
+19H → ?	+39H → P	+59H → ?
+1AH → 00H	+3AH → 00H	+5AH → ?
+1BH → A	+3BH → Q	+5BH → ?
+1CH → 00H	+3CH → 00H	+5CH → ?
+1DH → B	+3DH → R	+5DH → ?
+1EH → 00H	+3EH → 00H	+5EH → ?
+1FH → C	+3FH → S	+5FH → ?

*0000～000FH のリストは 0060:006C～007BH のコピー

*001A～004DH のリストは 0060:2820H の指すアドレスからのコピー

*DA/UA の値は MS–DOS ワークエリアのものと同一

*CONFIG.SYS で追加したブロックデバイスによっては、DA/UA の値がセットされないものがある。

得られた DA/UA とデバイスとの対応はつぎのとおり。

DA/UA	対応するドライブ
50～53	320K バイトフロッピーディスク
70～73	640K バイトフロッピーディスク
80～83	SASI ハードディスク
90～93	1M バイトフロッピーディスク
A0～A6	SCSI ハードディスク

機種別仕様一覧

- 項目名はつぎのようにに略す。また○は標準装備、△はオプション、×はそのハードウェアを取り付けられないことを示す。

CLK　動作クロック（MHz）
T-VRAM　テキストVRAM（Kバイト）
外字　ユーザー定義文字（文字）
G-VRAM　グラフィックVRAM（Kバイト）
DP　デュアルポートRAM
GC　グラフィックチャージャー
FM　FM音源

機種名	CPU	CLK	T-VRAM	外字	G-VRAM	DP	GC	FM
PC-9801	8086	5	8+(4)	×	96	×	×	△
PC-9801E	8086	5,8	8+(4)	63	96×2	×	×	△
PC-9801F	8086	5,8	12	63	96×2	×	×	△
PC-9801M	8086	5,8	12	63	96×2	×	×	△
PC-9801U2	V30	8	12	63	96(+32)	×	GRCG	△
PC-98XA	80286	8	12	188	128×4	×	GRCG	×
PC-9801VF2	V30	8	12	188	96(+32)×2	×	GRCG	△
PC-9801VM2	V30	8,10	12	188	96(+32)×2	×	GRCG	△
PC-9801UV2	V30	8,10	12	188	128×2	○	GRCG	○
PC-98LT	V50	8	—	188	32	×	×	×
PC-9801VM21	V30	8,10	12	188	128×2	○	GRCG	△
PC-9801VX2	V30/80286	8,10/8	12	188	128×2	○	EGC	△
PC-98XL（ノーマル）	V30/80286	8,10/8	12	188	128×2	○	EGC	△
PC-98XL（ハイレゾ）	80286	8	12	188	128×4	○	EGC	△
PC-9801UV21	V30	8,10	12	188	128×2	○	GRCG	○
PC-9801VX21	V30/80286	8,10/8,10	12	188	128×2	○	EGC	△
PC-98XL^2（ノーマル）	V30/80386	8/16	12	188	128×2	○	EGC	△
PC-98XL^2（ハイレゾ）	80386	16	12	188	128×4	○	EGC	△
PC-98LT21	V50	8	—	188	32	×	×	×
PC-9801UX21	V30/80286	8/10	12	188	128×2	○	EGC	○
PC-9801LV21	V30	8,10	12	188	128×2	○	GRCG	△
PC-9801CV21	V30	8,10	12	188	128×2	○	GRCG	○
PC-9801UV11	V30	8,10	12	188	128×2	○	GRCG	○
PC-9801RA2	V30/80386	8/16	12	188	128×2	○	EGC	△
PC-9801RX2	80286	8/10,12	12	188	128×2	○	EGC	△
PC-98LT22	V50	8	—	188	32	×	×	×
PC-9801LS2	80386	8/16	12	188	128×2	○	EGC	×
PC-9801VM11	V30	8,10	12	188	128×2	○	GRCG	△
PC-98RL2（ノーマル）	80386	8/20,16	12	188	128×2	○	EGC	△
PC-98RL2（ハイレゾ）	80386	20,16	12	188	128×4	○	EGC	△
PC-9801EX2	80286	8/10,12	12	188	128×2	○	EGC	○
PC-9801ES2	80386SX	8/16	12	188	128×2	○	EGC	△
PC-9801LX2	80286	8/10,12	12	188	128×2	○	EGC	△
PC-98DO	V30	8,10	12	188	128×2	○	GRCG	○
PC-9801LX5C	80286	8/10,12	12	188	128×2	○	EGC	△
PC-9801RA21	80386	8/16,20	12	188	128×2	○	EGC	△
PC-9801RS21	80386SX	8/16	12	188	128×2	○	EGC	△
PC-9801RX21	80286	8/10,12	12	188	128×2	○	EGC	△
PC-9801N	V30	10	12	188	128×2	○	GRCG	△
PC-9801NS	80386SX	12	12	188	128×2	○	EGC	△
PC-98RL21（ノーマル）	80386	8/20,16	12	188	128×2	○	EGC	△
PC-98RL21（ハイレゾ）	80386	20,16	12	188	128×4	○	EGC	△
PC-9801TW2	80386SX	8/20	12	188	128×2	○	EGC	△
PC-9801TS5	80386SX	8/20	12	188	128×2	○	EGC	△
PC-9801TF5	80386SX	8/20	12	188	128×2	○	EGC	△
PC-98DO+	V33	16,8	12	188	128×2	○	EGC	○
PC-9801NV	V30	16,8	12	188	128×2	○	GRCG	△
PC-9801DX2	80286	12,10	12	188	128×2	○	EGC	○
PC-9801UF	V30HL	16,8	12	188	128×2	○	GRCG	○
PC-9801UR	V30HL	16,8	12	188	128×2	○	GRCG	○

機種名	CPU	CLK	T-VRAM	外字	G-VRAM	DP	GC	FM
PC-9801DA	80386	20,16	12	188	128 × 2	○	EGC	○
PC-9801DS	80386SX	16	12	188	128 × 2	○	EGC	○
PC-9801CS	80386SX	16,8	12	188	128 × 2	○	EGC	○
PC-9801NS/E	80386SX	16	12	188	128 × 2	○	EGC	×
PC-9801NC	80386SX	20	12	188	128 × 2	○	EGC	×
PC-9801FA	80486SX	16	12	188	128 × 2	○	EGC	○

索　引

Y

Z

数字記号

ア

カ

マ

ヤ

ラ

ワ

■著者略歴

小高輝真（こだかてるまさ）
『The BASIC』（技術評論社刊）、『月刊アスキー』（アスキー出版局刊）などに記事を執筆。現在ウェブテクノロジ勤務。デバイスドライバ、BIOS等を主に開発している。日経MIXではpc9800会議の議長を務める。
日経MIX kodakoda／ASCII-NET pcs29576／PC-VAN JHF31202／NIFTY-Serve PDF01721

清水和文（しみずかずふみ）
学生時代から、通信ソフトウェアやデバイスドライバを作成。高機能な通信ソフトウェアとして有名なCCT-98IIIのチーフプログラマを勤める。また、『The BASIC』（技術評論社刊）などに執筆している。現在、ウェブテクノロジ勤務。
日経MIX cancer

速水　祐（はやみ　ゆう）
某学校で物理を教えるかたわら、コンピュータグループZOBplusで派手な活動を展開している。現在、『The BASIC』（技術評論社刊）にAdvanced Assemblerの連載を続けながらフリーソフトウェアの制作に忙しい毎日を費やす。著名なソフトウェアにRZ（ラムディスク）、VTZ（コンソールドライバ）、MSZ（マウスドライバ）などがある。
日経MIX hayami／ASCII-NET pcs19948

■執筆協力
山崎博義
國井　淳
古庄　歩
鈴木雅彦

■プログラム協力
鷲北　賢
西村克信

■編集協力
竹村章夫
田口景介

- 本書の内容に関するご質問は、小社第一書籍編集部まで、封書（返信用切手同封のこと）にてお願い致します。
電話によるお問い合わせには、応じられません。
なお、本書の範囲を越える質問に関しては、お答えできない場合もあります。
- 落丁・乱丁本は、送料当社負担にてお取り替え致します。
お手数ですが、小社営業部までご返送ください。

PC-9801 スーパーテクニック

1992年4月1日　初版発行
定価4,200円（本体4,078円）

著　者　小高 輝真/清水 和文/速水 祐
発行者　藤井 章生
編集人　佐藤 英一
発行所　**株式会社アスキー**
〒107-24　東京都港区南青山6-11-1スリーエフ南青山ビル
振　替　東京4-161144
大代表（03）3486-7111
出版営業部（03）3486-1977（ダイヤルイン）

制　作　株式会社 ジャパックス コミュニケーション システム
印　刷　大日本印刷 株式会社

編　集　中島 由弘/山下 憲治/高橋 正和/川島 良子

ISBN4-7561-0106-2　　　　Printed in Japan

● 13033

◀対象機種▶

PC-9800シリーズ（ノーマルモード）

関数の中には、PC-98HA/LTやハイレゾモードに対応しているもの、GRCGやEGCなどを搭載した機種にのみ対応しているものがあります。詳しくはAppendixの「ライブラリの動作環境」をお読みください。

◀ディスクの内容▶

●本書に掲載されたライブラリの全ソースファイル
●C言語から利用できる .LIBファイル
●ライブラリ関数を使ったサンプルプログラム

＊本ディスクにはMS-DOSや処理系は含まれていません。

◀必要な処理系▶

このディスクには、実行形式のファイルは含まれませんのでご注意ください。実行形式のファイルの作成、またはライブラリの再構築には、以下のアセンブラとC言語処理系が各1つ必要です。

●アセンブラ………MASM 5.1以上またはTASM 2.0以上
●C言語……………MS-C 6.0、Quick C 2.0、Borland C++ 2.0、Turbo C++ 2.0

◀利用条件▶

このディスクに収録されたプログラムは次のような条件で利用できます。

●プログラムを私的利用の範囲において使用すること
●プログラムを私的利用の範囲において、内容を変更、翻案および複製すること

また、以下のような事項を禁止します。

●本ディスクに収められたプログラムの全部または一部を、譲渡したり貸与すること
●プログラムに表示されている著作権その他権利者の表示を削除したり変更を加えること

なお、本書に掲載されたプログラムの使用結果については一切責任を負いかねますのでご了承ください。

PC-9801 スーパーテクニック

PC-9801のパワーを
120％引き出すテクニックを満載！

◆添付ディスクの内容および利用上の注意

■内　容　掲載されたテクニックをすぐに利用できるライブラリとソースコード

■対象機種　PC-9800シリーズ（ノーマルモード）
ただし、プログラムの中には、ハイレゾモードやPC-98LT/HAに対応しているもの、特定の機種では利用できないものがあります。具体的な利用可能機種については本文を参照してください。

■対象OS　MS-DOS 2.*xx* 以上

■対象処理系　プログラムはアセンブラもしくはC言語で記述されています。利用可能な処理系は、アセンブラがMASM 5.1以上、TASM 2.0以上。C言語が、QuickC 2.0、MS-C 6.0、Turbo C++ 2.0、Borland C++ 2.0です。

定価4,200円（本体4,078円）

ISBN4-7561-0106-2 C3055 P4200E